COMPARATIVE STUDY OF ORACLE SPATIAL AND POSTGRES SPATIAL

_____

A Thesis

Presented to the

Faculty of

San Diego State University

_____

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

_____

by

Shamal Kiran Matty

Spring 2012

SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Shamal Kiran Matty:

Comparative Study of Oracle Spatial and Postgres Spatial

Carl Eckberg

Carl Eckberg, Chair
Department of Computer Science

Joseph Lew

Joseph Lewis
Department of Computer Science

Robert Grone

Robert Grone
Department of Mathematics and Statistics

Feb 21, 2012

Approval Date

Copyright © 2012
by
Shamal Kiran Matty
All Rights Reserved

DEDICATION

I dedicate this thesis first and foremost to my husband Kiran for all his support and patience every step of the way. To all my family members, thank you all for the encouragement.

ABSTRACT OF THE THESIS

Comparative Study of Oracle Spatial and Postgres Spatial
by
Shamal Kiran Matty
Master of Science in Computer Science
San Diego State University, 2012

The primary focus of this thesis is Spatial Databases. This thesis shall cover a comprehensive study and comparison of two major databases namely Oracle and Postgres, which are extensively used in the handling of spatial data.

TABLE OF CONTENTS

PAGE

ABSTRACT v
LIST OF TABLES ix
LIST OF FIGURES x
LIST OF ACRONYMS xii
ACKNOWLEDGEMENTS xiii
CHAPTER
1 SPATIAL DATABASES 1
1.1 Introduction to Spatial Data 1
1.2 Geometry Types 1
1.3 Data Model 2
1.3.1 Element 2
1.3.2 Geometry 2
1.3.3 Layer 2
1.3.4 Coordinate System 3
1.4 Query Model 3
2 ADDING SPATIAL SUPPORT TO ORACLE AND POSTGRESQL 4
2.1 Oracle Spatial 4
2.2 PostGIS 4
3 SPATIAL DATA TYPES AND METADATA 5
3.1 Oracle Spatial Data Type 5
3.1.1 SDO_GEOMETRY Object Type 5
3.1.2 Geometry Metadata 6
3.1.3 Simple Example: Inserting Spatial Data 6
3.2 PostGIS Spatial Data Type 9
3.2.1 Spatial Metadata in PostGIS 9
3.2.2 Simple Example for Inserting Spatial Data in PostGIS 9
4 SPATIAL INDEXING 11

4.1 Indexing Algorithms in Oracle Spatial 11
4.1.1 R- Tree Indexing 11
4.1.2 Quad-Tree Indexing 12
4.1.3 Choosing R-Tree or Quad-Tree Indexing 13
4.2 Indexing in PostGIS 13
5 SPATIAL OPERATORS 15
5.1 Oracle Spatial Operators 15
5.2 PostGIS Spatial Operators 17
6 GEOMETRY PROCESSING FUNCTIONS 20
6.1 Buffering Functions 20
6.1.1 Buffering Function in Oracle Spatial 20
6.1.2 Buffering Function in PostGIS 20
6.2 Geometry Relationship Functions 21
6.2.1 Oracle Spatial Relationship Functions 21
6.2.2 PostGIS Relationship Functions 21
6.3 Geometry Combination Functions 22
6.3.1 Geometry Combination Functions in Oracle 22
6.3.2 PostGIS Geometry Combination Functions 22
6.4 Geometry Analysis functions 22
6.4.1 Area and Length Functions 23
6.4.2 Miscellaneous Geometry Analysis Functions 24
6.5 Aggregate functions 24
6.5.1 Aggregate Functions in Oracle Spatial 24
6.5.2 PostGIS Aggregate Functions 24
7 NETWORK MODELING 26
7.1 Oracle Network Modeling 26
7.1.1 Network Modeling Concepts 26
7.1.2 DATA Structures: The Network Tables 27
7.1.3 Network Data Model Application Programming Interface 27
7.1.4 Network Editor 28
7.2 Network Modeling in PostGIS 29
8 LOADING AND DISPENSING SPATIAL DATA 31

8.1 Loading and Dispensing Spatial Data in Oracle 31
8.1.1 Loading Shape Files 31
8.1.2 Dispensing Spatial Data to Shape Files 31
8.2 Loading and Dispensing Spatial Data in PostGIS 31
8.2.1 Loading Shape Files in PostGIS 31
8.2.2 Dispensing Spatial Data to Shape Files 31
9 VIEWING SPATIAL DATA 33
9.1 Oracle MapViewer 33
9.2 Viewing Spatial Data in PostGIS 33
10 CONCLUSION 35
11 FUTURE WORK 37
REFERENCES 38
APPENDIX
POSTGIS USER MANUAL 40

LIST OF TABLES

PAGE

Table 3.1. Values for T and D.....5
Table 4.1. Indexing Interpretations.....13
Table 4.2. R-Tree and Quad-Tree Indexing.....14
Table 5.1. Spatial Operators.....16
Table 5.2. PostGIS Spatial Operators.....18
Table 6.1. Oracle Spatial Relationship Functions.....21
Table 6.2. PostGIS Spatial Relationship Functions.....21
Table 6.3. Geometry Combination Functions in Oracle Spatial.....22
Table 6.4. Geometry Combination Functions in PostGIS.....23
Table 6.5. Area and Length Functions in Oracle Spatial.....23
Table 6.6. PostGIS Area and Length Functions.....23
Table 6.7. Miscellaneous Geometry Analysis Functions in Oracle Spatial.....24
Table 6.8. PostGIS Miscellaneous Geometry Analysis Functions.....24
Table 6.9. Aggregate Functions in Oracle Spatial.....25
Table 6.10. PostGIS Aggregate Functions.....25

LIST OF FIGURES

PAGE

Figure 1.1. Geometry types.....2
Figure 1.2. Query model. 3
Figure 3.1. Areas of interest for the example.....6
Figure 4.1. R- tree hierarchical index on MBR. 12
Figure 4.2. Fixed-size tiling at level 1. 12
Figure 4.3. Tesellated layer with multiple objects. 13
Figure 5.1. Overlap operator output.....18
Figure 5.2. Overlap to the right operator output. 19
Figure 6.1. Distance buffers for points, lines, and polygons. 20
Figure 6.2. Results of union, intersection, difference, xor.....22
Figure 7.1. Network tables. 27
Figure 7.2. Simple spatial network. 28
Figure 7.3. Shortest path from node 1 to node 3. 29
Figure 7.4. Shortest route through the Twin Cities. 30
Figure 8.1. Using shp2pgsql-gui to load the bc_pubs table. 32
Figure 9.1. Map viewer showing the cola_markets table. 33
Figure 9.2. QGIS showing the cola_market table. 34
Figure A.1. Welcome page. 41
Figure A.2. Installation directory. 42
Figure A.3. Installation directory. 42
Figure A.4. Password. 43
Figure A.5. Port number. 43
Figure A.6. Advanced options. 44
Figure A.7. Ready to install. 44
Figure A.8. Installing. 45
Figure A.9. Setup complete. 45
Figure A.10. Welcome to stack builder. 46

Figure A.11. Select category. 46
Figure A.12. Select mirror site. 47
Figure A.13. Selected packages. 47
Figure A.14. Downloading. 48
Figure A.15. Installation files downloaded. 48
Figure A.16. License agreement. 49
Figure A.17. Choose components. 49
Figure A.18. Choose install location 50
Figure A.19. Database connection. 50
Figure A.20. Installing PostGIS. 51
Figure A.21. Installation complete 51
Figure A.22. pgAdmin III. 52
Figure A.23. New database creation. 53
Figure A.24. Tables with PostGIS installed 54
Figure A.25. Query to create table. 55
Figure A.26. Shape file to PostGIS importer. 58
Figure A.27. Add a new connection. 59
Figure A.28. Create a new PostGIS connection. 60
Figure A.29. Enter credentials. 61
Figure A.30. Select PostGIS table. 61
Figure A.31. cola_market table 62
Figure A.32. bc_pubs layer. 63

LIST OF ACRONYMS

SQL	Structured Query Language
GIS	Geographic Information System
GPL	General Public License
CAD	Computer Aided Design
OGC	Open GIS Consortium
SRID	Spatial Reference ID
WKT	Well-Known Text
MBR	Minimum Bounding Rectangle
GiST	Generalized Search Tree
PL/SQL	Procedural Language/Structured Query Language
API	Application Programming Interface
ESRI	Environmental Systems Research Institute
QGIS	Quantum GIS
JUMP	Java Unified Mapping Platform
UDIG	User-Friendly Desktop Internet GIS
FME	Feature Manipulation Engine
OGR	OpenGIS Simple Features Reference Implementation
GRASS	Geographic Resources Analysis Support System

ACKNOWLEDGEMENTS

Thank you to my committee – Professors Eckberg, Lewis and Grone. Thanks especially to Professor Eckberg, the chair of my committee, for all your guidance, inspiration and encouragement.

CHAPTER 1

SPATIAL DATABASES

A Spatial Database is a database that defines special data types for geometric objects and allows storage of geometric data (usually of a geographic nature) in database tables [1]. It provides special functions and indexes for querying and manipulating that data using extensions of Structured Query Language (SQL).Oracle Spatial and PostGIS (Postgres Spatial) are examples of commonly used spatial databases.

1.1 INTRODUCTION TO SPATIAL DATA

Spatial data is the information that identifies the geographic location of features and boundaries on earth, such as natural or constructed features, oceans, and more. It is usually stored as coordinates and topology, and is data that can be mapped.

A common example of spatial data can be seen in a road map. A road map is a two-dimensional object that contains points, lines, and polygons that can represent cities, roads, and political boundaries such as states or provinces. A road map is a visualization of geographic information. The location of cities, roads, and political boundaries that exist on the surface of the Earth are projected onto a two-dimensional display or piece of paper, preserving the relative positions and relative distances of the rendered objects.

1.2 GEOMETRY TYPES

A geometry is an ordered sequence of vertices that are connected by straight line segments or circular arcs. The semantics of the geometry are determined by its type. Spatial supports several primitive types, and geometries composed of collections of these types, including two-dimensional. All those types are represented by points, or points and edges/arcs, and are a “vector” representation. The other common representation is “raster”, e.g. satellite photos. Figure 1.1 depicts the different geometries.

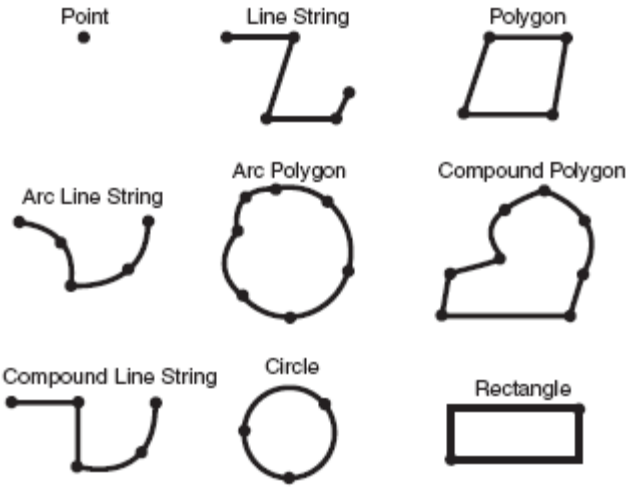

Figure 1.1. Geometry types.

1.3 DATA MODEL

The spatial data model is a hierarchical structure consisting of elements, geometries, and layers. Layers are composed of geometries, which in turn are made up of Elements.

1.3.1 Element

An element is the basic building block of a geometry. The supported spatial element types are points, line strings, and polygons. Point data consists of one coordinate. Line data consists of two coordinates representing a line segment of the element. Polygon data consists of coordinate pair values, one vertex pair for each line segment of the polygon.

1.3.2 Geometry

A geometry (or geometry object) is the representation of a spatial feature, modeled as an ordered set of primitive elements. A geometry can consist of a single element, which is an instance of one of the supported primitive types, or a homogeneous or heterogeneous collection of elements.

1.3.3 Layer

A layer is a collection of geometries that have the same attribute set. For example, one layer in GIS might include topographical features, while another describes population density.

1.3.4 Coordinate System

A coordinate system (a.k.a. a spatial reference system) is a means of assigning coordinates to a location and establishing relationships between sets of such coordinates. Spatial data can be associated with a Cartesian, geodetic (geographical), projected, or local coordinate system:

- Cartesian coordinates are coordinates that measure the position of a point from a defined origin along axes that are perpendicular in the represented two-dimensional or three-dimensional space. If a coordinate system is not explicitly associated with geometry, a Cartesian coordinate system is assumed.
- Geodetic coordinates (sometimes called geographic coordinates) are angular coordinates (longitude and latitude), closely related to spherical polar coordinates, and are defined relative to a particular Earth geodetic datum. (A geodetic datum is a means of representing the figure of the Earth and is the reference for the system of geodetic coordinates.)
- Projected coordinates are planar Cartesian coordinates that result from performing a mathematical mapping from a point on the Earth's surface to a plane. There are many such mathematical mappings, each used for a particular purpose.
- Local coordinates are Cartesian coordinates in a non-Earth (non-georeferenced) coordinate system. Local coordinate systems are often used for CAD applications and local surveys.

1.4 QUERY MODEL

A spatial database uses a two-tier query model to resolve spatial queries and spatial joins. It is used to indicate that two distinct operations are performed to resolve queries. The output of the two combined operations yields the exact result set. A spatial database uses a spatial index to implement the primary filter. Figure 1.2 depicts the query model.

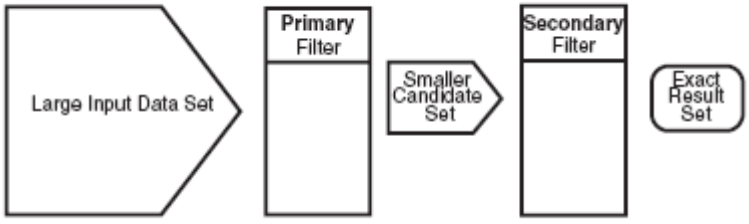

Figure 1.2. Query model.

CHAPTER 2

ADDING SPATIAL SUPPORT TO ORACLE AND POSTGRESQL

This chapter shall provide details on how to spatially enable Oracle and PostgresSQL object-relational databases.

2.1 ORACLE SPATIAL

Oracle Spatial forms a separately-licensed option component of the Oracle Database. It helps users manage geographic and location-data in a native type within an Oracle database. It is a proprietary closed source that comes packaged with Oracle Enterprise edition which can be downloaded from [2]. It is free to “students”.

2.2 POSTGIS

PostGIS adds support for geographic objects to the PostgreSQL object-relational database. In effect, PostGIS "spatially enables" the PostgreSQL server, allowing it to be used as a backend spatial database for Geographic Information Systems (GIS). It is a GPL open source that can be installed as an add-in to the PostgreSQL database. The PostgreSQL database can be downloaded from [3]. Refer to the Appendix for details.

CHAPTER 3

SPATIAL DATA TYPES AND METADATA

A spatial database supports various data types to process spatial data. It provides various operators, functions and procedures that make use of these types.

3.1 ORACLE SPATIAL DATA TYPE

Spatial data is specified using two components: a location with respect to some origin and a geometric shape. Oracle Spatial provides SDO_GEOMETRY data type that captures the location and shape information of data rows in a table. This data type is internally represented as an Oracle object data type and conforms to the Open GIS Consortium (OGC) Geometry model [4].

3.1.1 SDO_GEOMETRY Object Type

The SDO_GEOMETRY object type consists of following attributes:

- SDO_GTYPE - This attribute captures what type of geometry is being represented.
- It is a four-digit number structured as D00T. Table 3.1 specifies the values for D and T based on dimension and shape of the geometry.

Table 3.1. Values for T and D

Digit	Values
D (dimension)	2,3,4
T=0	Uninterpreted type
T=1	Point
T=2	Line
T=3	Polygon
T=5	Multipoint
T=6	Multiline
T=7	Multipolygon
T=4	Collection

- SDO_SRID - This specifies the spatial reference system in which the location/shape of geometry is specified. For example, NULL means local coordinates, 8307 is standard lat/long.

- SDO_POINT - If the geometry is a point, then the coordinates of geometry can be stored in this attribute, else it is null.
- SDO_ORDINATES - It stores the coordinates of all elements of the geometry.
- SDO_ELEM_INFO – It specifies where in the SDO_ORDINATES array a new element starts, how it is connected and whether it is a point, line, or a polygon.

3.1.2 Geometry Metadata

The geometry metadata is the data that describes the lower and upper bound of the geometry dimensions and the tolerance in each dimension. It is stored in a global table owned by MDSYS. Each spatial user has a view named USER_SDO_GEOM_METADATA available in the schema associated with that user. Spatial users are responsible for populating these views with metadata information for tables owned by users.

3.1.3 Simple Example: Inserting Spatial Data

The scenario consists of a soft drink manufacturer who has identified geographical areas of marketing interest for several products (colas), which could be those produced by the company or by its competitors, or some combination [1]. Each area of interest could represent any user-defined criterion: for example, an area where that cola has either the majority market share or is under competitive pressure, or is believed to have significant growth potential. Each area could be a neighborhood in a city, or a part of a state, a province, or a country. Figure 3.1 shows the areas of interest for the four colas. This is a standard Oracle spatial example.

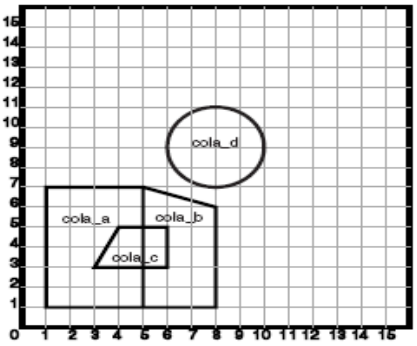

Figure 3.1. Areas of interest for the example.

Example 3.1:

```
CREATE TABLE cola_markets (
mkt_id NUMBER PRIMARY KEY,
name VARCHAR2(32),
shape SDO_GEOMETRY);
INSERT INTO cola_markets VALUES (
1,
'cola_a',
SDO_GEOMETRY (
2003, -- two-dimensional polygon
NULL,
NULL,
SDO_ELEM_INFO_ARRAY (1, 1003, 3), -- one rectangle (1003 = exterior)
SDO_ORDINATE_ARRAY (1, 1, 5, 7) -- only 2 points needed to
-- define rectangle (lower left and upper right) with
-- Cartesian-coordinate data
)
);
INSERT INTO cola_markets VALUES (
2,
'cola_b',
SDO_GEOMETRY (
2003, -- two-dimensional polygon
NULL,
NULL,
SDO_ELEM_INFO_ARRAY (1, 1003, 1), -- one polygon (exterior polygon ring)
SDO_ORDINATE_ARRAY (5, 1, 8, 1, 8, 6, 5, 7, 5, 1)
)
);
INSERT INTO cola_markets VALUES (
3,
```

```
'cola_c',
SDO_GEOMETRY (
2003, -- two-dimensional polygon
NULL,
NULL,
SDO_ELEM_INFO_ARRAY (1, 1003, 1), -- one polygon (exterior polygon ring)
SDO_ORDINATE_ARRAY (3, 3, 6, 3, 6, 5, 4, 5, 3, 3)
)
);
INSERT INTO cola_markets VALUES (
4,
'cola_d',
SDO_GEOMETRY (
2003, -- two-dimensional polygon
NULL,
NULL,
SDO_ELEM_INFO_ARRAY (1, 1003, 4), -- one circle
SDO_ORDINATE_ARRAY (8, 7, 10, 9, 8, 11)
)
);
--Populate the metadata table
INSERT INTO user_sdo_geom_metadata
(TABLE_NAME,
COLUMN_NAME,
DIMINFO,
SRID)
VALUES (
'cola_markets',
'shape',
SDO_DIM_ARRAY (-- 20X20 grid
SDO_DIM_ELEMENT ('X', 0, 20, 0.005),
```

```
SDO_DIM_ELEMENT ('Y', 0, 20, 0.005)
),
NULL -- SRID
);
------CREATE THE SPATIAL INDEX --
CREATE INDEX cola_spatial_idx
ON cola_markets (shape)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;
-- Preceding statement created an R-tree index.
```

3.2 POSTGIS SPATIAL DATA TYPE

PostGIS uses geometry type to store spatial data in a table. It creates geometries from OGC Well-known Text (WKT) using the function ST_GeomFromText [5]. This function also needs to know the spatial reference system (SRID) of the geometry. Once the table with geometries is created, the spatial user is responsible for populating the geometry metadata using the Populate_Geometry_Columns function [6]. This approach is used to store geometries of different types in a single table. Refer to the Appendix for details.

Alternatively, a spatial column could be added to the table using the OpenGIS "AddGeometryColumn" function [6].

3.2.1 Spatial Metadata in PostGIS

PostGIS uses a table named geometry_columns to store metadata associated with the geometry columns in the database. The installation of PostGIS automatically creates this table. The geometry_columns table provides housekeeping information about geometry columns in the database and is commonly used by third-party tools to gather a list of geometry layers in the database.

3.2.2 Simple Example for Inserting Spatial Data in PostGIS

The scenario used to create tables here is the same as described in Section 3.1.3. Example 3.2 creates a table and inserts spatial data in it.

Example 3.2:

```
CREATE TABLE cola_market (
mkt_id int2,
name VARCHAR (32),
geom GEOMETRY);
INSERT INTO cola_market (mkt_id, name, geom)
VALUES (1,'cola_a', ST_GeomFromText (‘POLYGON ((1 1, 1 7, 5 1, 1 1))',-1));
INSERT INTO cola_market (mkt_id, name, geom)
VALUES (2,'cola_b', ST_GeomFromText (‘POLYGON ((5 1, 5 7, 8 6, 8 1, 5 1))',-1));
INSERT INTO cola_market (mkt_id,name,geom)
VALUES (3,'cola_c', ST_GeomFromText (‘POLYGON ((3 3, 4 5, 6 5, 6 3, 3 3))',-1));
INSERT INTO cola_market (mkt_id,name,geom)
VALUES (4,'cola_d', ST_GeomFromText (‘CIRCULARSTRING (6 9, 10 9, 6 9)',-1));
```

CHAPTER 4

SPATIAL INDEXING

Indexing allows adequate query resolution speed for any DBMS. Because a DBMS is not memory resident a special data structure is used to speed searches. These are B-Trees. But the columns searched are still one dimensional.

Indexing is also what makes spatial database suitable for large data sets. In its absence, any search for a feature would entail "sequential scan" of every record in the database. Indexing speeds up searching by organizing the data into a search tree which can be quickly traversed to find a particular record. Because spatial is two or more dimensional, B-Trees alone are not adequate.

4.1 INDEXING ALGORITHMS IN ORACLE SPATIAL

Oracle spatial uses two special spatial indexing techniques, namely R- Tree and Quad-Tree. R- Tree is the default indexing algorithm. A spatial index must be created in Oracle spatial for a spatial table.

4.1.1 R- Tree Indexing

A spatial R-tree index can index spatial data of up to 4 dimensions [1]. An R-tree index approximates each geometry by a single rectangle that minimally encloses the geometry (called the minimum bounding rectangle, or MBR). For a layer of geometries, an R-tree index consists of a hierarchical index on the MBRs of the geometries in the layer, as shown in Figure 4.1

In Figure 4.1:

- 1 through 9 are geometries in a layer.
- a, b, c, and d are the leaf nodes of the R-tree index, and contain minimum bounding rectangles of geometries, along with pointers to the geometries. For example, a contains the MBR of geometries 1 and 2, b contains the MBR of geometries 3 and 4, and so on.

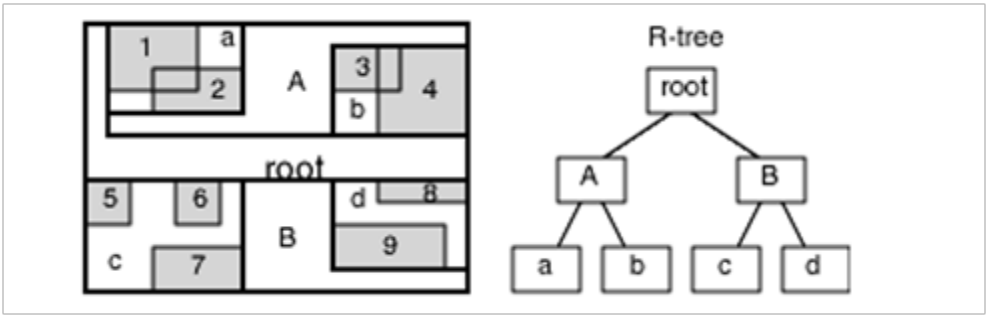

Figure 4.1. R- tree hierarchical index on MBR.

- A contains the MBR of a and b, and B contains the MBR of c and d.
- The root contains the MBR of A and B (that is, the entire area shown).

4.1.2 Quad-Tree Indexing

In the linear quadtree indexing scheme, the coordinate space (for the layer where all geometric objects are located) is subjected to a process called tessellation, which defines exclusive and exhaustive cover tiles for every stored geometry [7]. Tessellation is done by decomposing the coordinate space in a regular hierarchical manner. The range of coordinates, the coordinate space, is viewed as a rectangle. At the first level of decomposition, the rectangle is divided into halves along each coordinate dimension generating four tiles as shown in Figure 4.2. Each tile that interacts with the geometry being tessellated is further decomposed into four tiles. This process continues until some termination criteria, such as size of the tiles or the maximum number of tiles to cover the geometry, is met. The SDO_LEVEL value is used while tessellating objects. Increasing the level results in smaller tiles and better geometry approximations as depicted in Figure 4.3. Table 4.1 shows the different indexing interpretations.

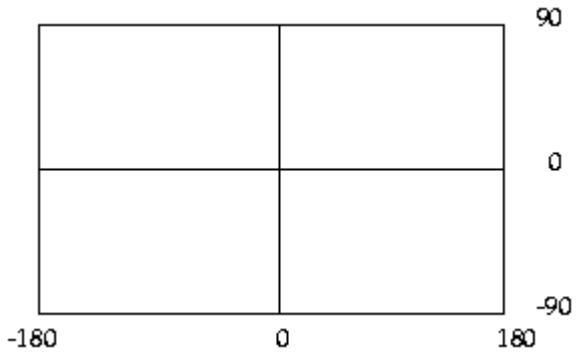

Figure 4.2. Fixed-size tiling at level 1.

Figure 4.3. Tesellated layer with multiple objects.

Table 4.1. Indexing Interpretations

SDO_LEVEL	SDO_NUMTILES	Action
Not specified or 0	Not specified or 0	R-tree index.
>= 1	Not specified or 0	Fixed indexing (indexing with fixed-size tiles).
>= 1	>= 1	Hybrid indexing with fixed-size and variable-sized tiles. The SDO_LEVEL column defines the fixed tile size. The SDO_NUMTILES column defines the number of variable tiles to generate per geometry.

4.1.3 Choosing R-Tree or Quad-Tree Indexing

Quad- Tree has advantages in terms of some more complex queries, but basic spatial operations are performed much faster using an R-Tree indexing type[8]. Table 4.2 shows when to choose R-tree and Quad-Tree indexing[1, 9].

4.2 INDEXING IN POSTGIS

PostGIS indexes are R-Tree indexes, implemented on top of the general GiST (Generalized Search Tree) indexing schema. The GiST is an extensible data structure, which allows users to develop indices over any kind of data, supporting any lookup over that data. This package unifies a number of popular search trees in one data structure (the long list of potentials includes R-trees, B+-trees, hB-trees, TV-trees, Ch-Trees, partial sum trees, ranked B+-trees, and many, many others), eliminating the need to build multiple search trees for

Table 4.2. R-Tree and Quad-Tree Indexing

	R-Tree Indexing	Quad –Tree Indexing
1.	The approximation of geometries cannot be fine tuned. (Spatial uses the minimum bounding rectangles.)	The approximation of geometries can be fine tuned by setting the tiling level and number of tiles.
2.	Index creation and tuning are easier.	Tuning is more complex, and setting the appropriate tuning parameter values can affect performance significantly.
3.	Less storage is required.	More storage is required.
4.	If your application workload includes nearest neighbor queries (SDO_NN operator), R-tree indexes are faster, and you can use the sdo_batch_size keyword.	If your application workload includes nearest neighbor queries (SDO_NN operator), quadtree indexes are slower, and you cannot use the sdo_batch_size keyword.
5.	Heavy update activity to the spatial column may decrease the R-tree index performance until the index is rebuilt.	Heavy update activity does not affect the performance of a quadtree index.
6.	You can index up to four dimensions.	You can index only two dimensions
7.	An R-tree index is recommended for indexing geodetic data if SDO_WITHIN_DISTANCE queries will be used on it	A quadtree index is not recommended for Indexing geodetic data if SDO_WITHIN_DISTANCE queries will be used on it.

handling diverse applications. In addition to unifying all these structures, the GiST has one key feature that previous trees lacked: both data and query extensibility. [10].

Default index type is the B-Tree. B-Tree indices are not lossy (inexact) in the way a GiST index can be. This means that while the GIST index only indexes the bounding box of the geometry, the B-Tree must index the entire geometry, which can often be larger than the index can cope with. Only the bounding-box-based operators such as && can take advantage of the GiST spatial index. Most of the geometry relationship operators include the implicit bounding box overlap operators.

CHAPTER 5

SPATIAL OPERATORS

Spatial operators are used to perform spatial analysis. They are tied to a spatial index, and in most cases evaluated in a two-stage filtering mechanism involving the spatial index. This evaluation using a spatial index is referred to as the primary filter. Here, the approximations in the index (the MBRs stored in the spatial index table) are used to identify a candidate set of rows that satisfies the operator relationship with respect to query location. The identified rows are then passed through the Geometry Engine, referred to as the secondary filter, to return the correct set of rows for that operator. All of this processing is transparent to the user.

5.1 ORACLE SPATIAL OPERATORS

Oracle provides different operators that can be used to perform spatial analysis. All the spatial operators follow a general syntax as described below:

<spatial operator>
(
Table_geometry IN SDO_GEOMETRY ,
query_geometry IN SDO_GEOMETRY
[, parameter_string IN VARCHAR2
[, tag IN NUMBER]]
)
= 'TRUE'

- Table_geometry - SDO_GEOMETRY column of the table on which the operator is applied.
- Query_geometry - Query location. This could be an SDO_GEOMETRY column of another table, a bind variable, or a dynamically constructed object.
- Parameter_string - Parameters specific to the operator. These are optional.
- Tag – number used only in specific operators.

Table 5.1 describes the main spatial operators [1].

Table 5.1. Spatial Operators

Operator	Description
SDO_FILTER	Specifies which geometries may interact with a given geometry.
SDO_JOIN	Performs a spatial join on one or more topological relationships.
SDO_NN	Determine the nearest neighbor geometries to a geometry.
SDO_RELATE	Determines whether or not two geometries interact in a specified way.
SDO_WITHIN_DISTANCE	Determines whether two geometries are within a specified distance of one another.

SDO_ANYINTERACT, SDO_CONTAINS, SDO_COVERS, SDO_COVEREDBY, SDO_EQUAL, SDO_INSIDE, SDO_ON, SDO_OVERLAPS, and SDO_TOUCH are simplified variants of the SDO_RELATE operator for specific type of interactions. See [1] for detailed information on each operator.

This section provides examples for few spatial operators.

Example 5.1: The following example finds the two objects from the SHAPE column in the COLA_MARKETS table that are nearest to a specified point (10, 7) [1]. (The example uses the definitions and data described in Section 3.1.3 and illustrated in Figure 3.1.)

```
SELECT /*+ INDEX(c cola_spatial_idx) */
c.mkt_id, c.name  FROM cola_markets c  WHERE SDO_NN(c.shape,
sdo_geometry(2001, NULL, sdo_point_type(10,7,NULL), NULL,
NULL),  'sdo_num_res=2') = 'TRUE';

  MKT_ID    NAME
---------- --------------------------------
       2          cola_b
       4          cola_d
```

Example 5.2: The following example (taken from [1]) shall find geometries that have the COVEREDBY relationship with a query window (here, a rectangle with lower-left, upper-right coordinates 1, 1, 5, 8). (This example uses the definitions and data described in Section 3.1.2 and illustrated in Figure 3.1)

```
SELECT c.mkt_id, c.name
FROM cola_markets c
WHERE SDO_COVEREDBY(c.shape,
        SDO_GEOMETRY(2003, NULL, NULL,
          SDO_ELEM_INFO_ARRAY(1,1003,3),
          SDO_ORDINATE_ARRAY(1,1, 5,8))
        ) = 'TRUE';
```

MKT_ID	NAME
1	cola_a

5.2 PostGIS Spatial Operators

Each geometry has a bounding box defined as the smallest rectangular box that completely encloses the geometry. PostGIS offers a number of geometry bounding box comparators that work exclusively with box2d objects and one comparator that works against the actual geometry. Some but not all of these operators have functional counterparts that apply to the entire geometry. As a convenient shorthand, PostGIS uses various operators to symbolize comparators. All PostGIS operators use a gist index except the “=” operator, which uses B-tree index [11]. Table 5.2 shows the PostGIS operators that can be applied to geometries.

This section shall provide examples [6] for few spatial operators that are used in PostGIS.

Example 5.3: The following example shall demonstrate the && operator as depicted in Figure 5.1.

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 && tbl2.column2 AS overlaps
FROM ( VALUES
        (1, 'LINESTRING(0 0, 3 3)'::geometry),
        (2, 'LINESTRING(0 1, 0 5)'::geometry)) AS tbl1,
        ( VALUES
```

Table 5.2. PostGIS Spatial Operators

Operator	What it checks
&&	Returns true if A's bounding box overlaps B's.
&<	Returns true if A's bounding box overlaps or is to the left of B's.
&<|	Returns true if A's bounding box overlaps or is to the left of B's.
&>	Returns true if A's bounding box overlaps or is to the right of B's
<<	Returns true if A's bounding box is strictly to the left of B's
<<|	Returns true if A's bounding box is strictly below B's
=	Returns true if A's bounding box is the same as B's
>>	Returns true if A's bounding box is strictly to the right of B's
@	Returns true if A's bounding box is contained by B's
|&>	Returns true if A's bounding box overlaps or is above B's
|>>	Returns true if A's bounding box is strictly above B's
~ -	Returns true if A's bounding box contains B's

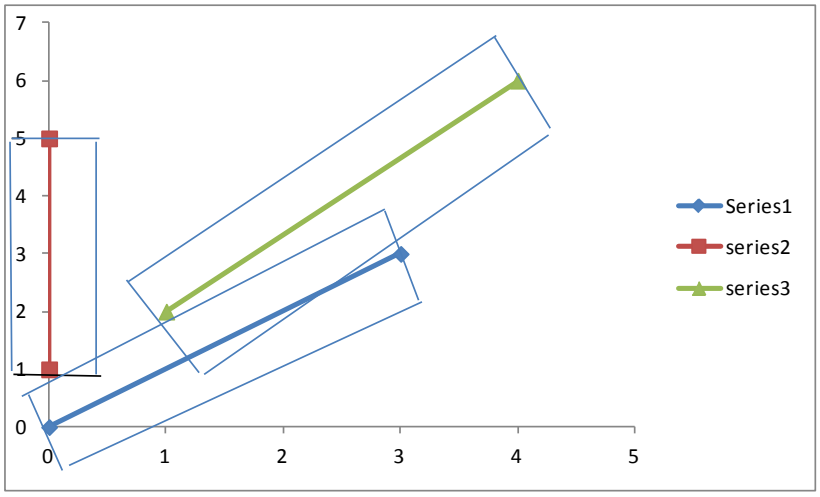

Figure 5.1. Overlap operator output.

(3, 'LINESTRING(1 2, 4 6)'::geometry)) AS tbl2;

column1 | column1 | overlaps

---------+---------+----------

1 | 3 | t

2 | 3 | f

Example 5.4: The following example shall demonstrate the &> operator as depicted in Figure 5.2.

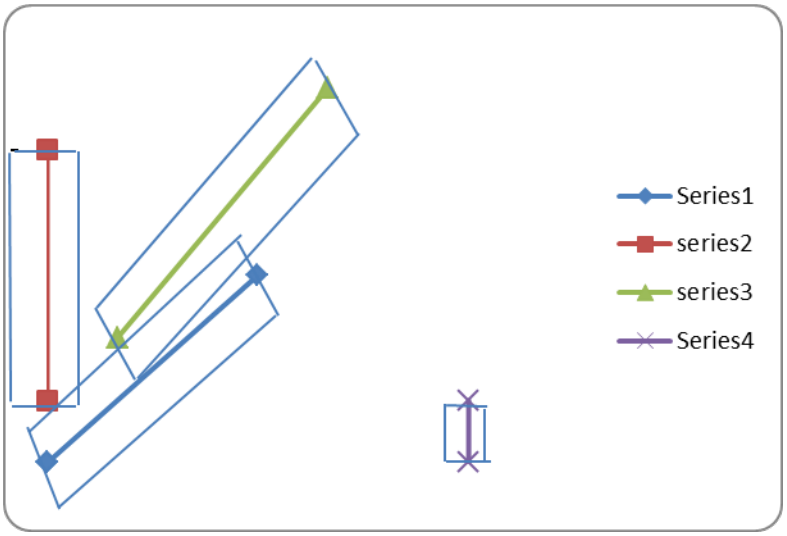

Figure 5.2. Overlap to the right operator output.

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &> tbl2.column2 AS overright
FROM
  ( VALUES
        (1, 'LINESTRING(1 2, 4 6)'::geometry)) AS tbl1,
  ( VALUES
        (2, 'LINESTRING(0 0, 3 3)'::geometry),
        (3, 'LINESTRING(0 1, 0 5)'::geometry),
        (4, 'LINESTRING(6 0, 6 1)'::geometry)) AS tbl2;
 column1 | column1 | overright
---------+---------+-----------
       1 |       2 | t
       1 |       3 | t
       1 |       4 | f
(3 rows)
```

CHAPTER 6

GEOMETRY PROCESSING FUNCTIONS

Geometry processing functions, which are also referred to as spatial functions, are used to perform spatial analysis. In contrast to spatial operators, these geometry processing functions

- Do not require a spatial index.
- Provide more detailed analyses than the spatial operators associated with a spatial index.
- Can appear in the SELECT list (as well as the WHERE clause) of a SQL statement.

6.1 BUFFERING FUNCTIONS

This function constructs a buffer around a specified geometric object or a set of geometric objects.

6.1.1 Buffering Function in Oracle Spatial

Oracle spatial provides a function SDO_BUFFER which generates a buffer polygon around or inside a geometry object [1]. Figure 6.1 illustrates the distance buffers.

Figure 6.1. Distance buffers for points, lines, and polygons.

The values for units of numerical distance to buffer the input geometry can be obtained by consulting MDSYS.SDO_DIST_UNITS table [1].

6.1.2 Buffering Function in PostGIS

PostGIS provides a function ST_Buffer to generate a buffer around a geometry [6]. Unlike Oracle, calculations are in the Spatial Reference System of the geometry. There is no choice of units.

6.2 GEOMETRY RELATIONSHIP FUNCTIONS

These functions are used to analyze the relationship between two geometry objects.

6.2.1 Oracle Spatial Relationship Functions

Oracle provides two relationship functions. Refer to [1] for details on these functions. Table 6.1 illustrates these functions.

Table 6.1. Oracle Spatial Relationship Functions

Function	Description
SDO_DISTANCE	Computes the minimum distance between any two points on the two geometries.
SDO_RELATE	Returns the type of relationship between the two geometry objects.

6.2.2 PostGIS Relationship Functions

The relationship functions in PostGIS do not support curved geometries [10]. To apply the relationship function to a curve, the curve needs to be approximated to a non-curve using function ST_CurveToLine [6]. PostGIS relationship functions will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries. To avoid index use, use the function name preceded by a _.Table 6.2 [12] shows the different spatial relationship functions in PostGIS.

Table 6.2. PostGIS Spatial Relationship Functions

Function	Description
ST_Distance	Returns Cartesian distance between two geometries. Does not use spatial GiST indexes.
ST_DWithin	Returns true if geometries are within the specified distance of one another. Uses GiST indexes if available.
ST_Equals	Returns 1 (TRUE) if the given Geometries are "spatially equal".
ST_Disjoint	Returns 1 (TRUE) if the Geometries are "spatially disjoint".
ST_Intersects	Returns 1 (TRUE) if the Geometries are "spatially intersect".
ST_Relate	Returns the DE-9IM (dimensionally extended nine-intersection matrix).

Source: WIKIPEDIA, *DE-9IM*. Wikipedia, http://en.wikipedia.org/wiki/DE-9IM, accessed, November 2011, 2011.

ST_Touches, ST_Overlaps, ST_Crosses, ST_Contains, ST_Covers, ST_Coveredby functions are also available that test for specific type of relationship between the geometries.

6.3 GEOMETRY COMBINATION FUNCTIONS

The geometry combination functions operate on a pair of geometries. If geom1 and geom2 are two geometries, the semantics of these functions are illustrated in Figure 6.2.

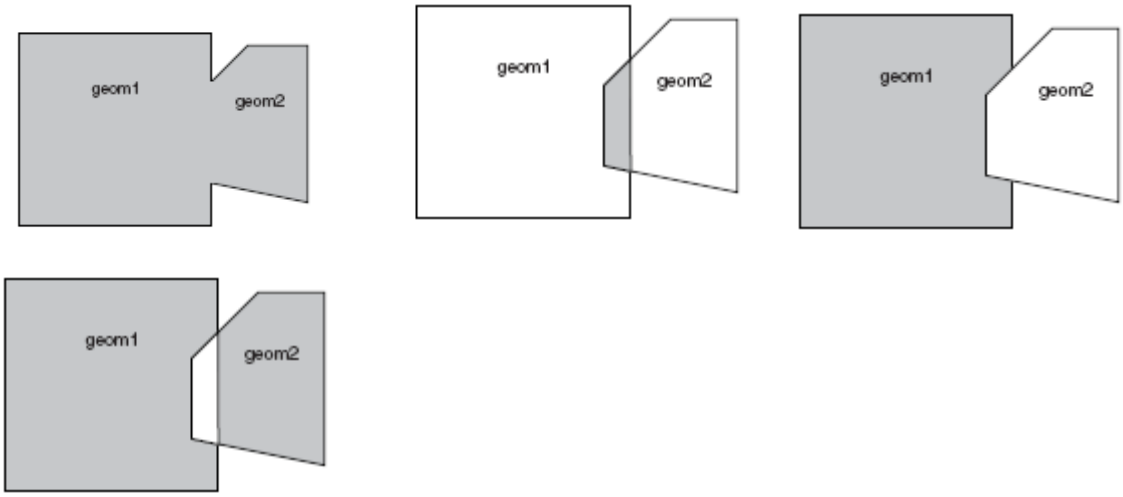

Figure 6.2. Results of union, intersection, difference, xor.

6.3.1 Geometry Combination Functions in Oracle

Oracle spatial provides four geometry combination functions as shown in Table 6.3 [1]. Refer to [1] for details on each of these functions.

Table 6.3. Geometry Combination Functions in Oracle Spatial

Function	Description
SDO_UNION	Returns the region covered by A or B.
SDO_INTERSECTION	Returns the region of A that is also shared by B.
SDO_DIFFERENCE	Returns the region covered by A that is not also covered by B.
SDO_XOR	Returns the region of A and B not shared by both.

Source: C. MURRAY, *Oracle spatial developer's guide 11g release 1 (11.1)*, Oracle, Redwood Shores, California, 2006.

6.3.2 PostGIS Geometry Combination Functions

PostGIS offers various geometry combination functions equivalent to the functions in Oracle Spatial [10]. Table 6.4 illustrates these functions.

6.4 GEOMETRY ANALYSIS FUNCTIONS

This section provides details on the geometry analysis functions.

Table 6.4. Geometry Combination Functions in PostGIS

Function	Description
ST_UNION	Returns the region covered by A or B.
ST_INTERSECTION	Returns the region of A that is also shared by B.
ST_DIFFERENCE	Returns the region covered by A that is not also covered by B.
ST_SYMDIFFERENCE	Returns the region of A and B not shared by both.

6.4.1 Area and Length Functions

Both Oracle spatial and PostGIS offer various area and length function. The main difference is that Oracle allows users to select the units for measurement. Table 6.5 shows the area and length functions in Oracle Spatial.

Table 6.5. Area and Length Functions in Oracle Spatial

Function	Description
SDO_AREA	Returns the area of geometry object. The units for measurement are chosen from table MDSYS.SDO_AREA_UNITS.
SDO_LENGTH	Returns the length for a line string, perimeter for a polygon and zero for points

In PostGIS, the units for measurement are the units of the spatial reference system under consideration. PostGIS also offers functions which take the earth's curvature into consideration to provide more accurate results. Table 6.6 illustrates these functions [10].

Table 6.6. PostGIS Area and Length Functions

Function	Description
ST_AREA	Returns the area of geometry if it is a polygon or multi-polygon.
ST_LENGTH	Returns the 2d length of the geometry if it is a linestring or multilinestring.
ST_PERIMETER	Returns the perimeter of geometry if it is a polygon or multi-polygon.
ST_DISTANCE	Returns the 2-dimensional Cartesian minimum distance between two geometries in projected units.
ST_DISTANCE_SPHEROID	Returns linear distance between two lon/lat points given a particular spheroid. Currently only implemented for points.
ST_LENGTH_SPHEROID	Calculates the length of a linestring/multilinestring on an ellipsoid.

6.4.2 Miscellaneous Geometry Analysis Functions

These functions perform simple geometric analyses on a single geometry object. Table 6.7 shows the miscellaneous functions used in Oracle spatial [1].

Table 6.7. Miscellaneous Geometry Analysis Functions in Oracle Spatial

Function	Description
SDO_CONVEXHULL	Returns the smallest convex polygon that completely encloses the geometry.
SDO_CENTROID	Returns the geometric center of geometry.

PostGIS offers a large number of geometry analyses functions as compared to Oracle Spatial. Table 6.8 shows the main miscellaneous functions used in PostGIS [10].

Table 6.8. PostGIS Miscellaneous Geometry Analysis Functions

Function	Description
ST_CONVEXHULL	Returns the smallest convex polygon that completely encloses the geometry.
ST_CENTROID	Returns the geometric center of geometry.
ST_POINTONSURFACE	Returns a point that is guaranteed to intersect a surface.
ST_GEOMETRYTYPE	Returns the type of the geometry as a string.
ST_NUMPOINTS	Return the number of points in a ST_LineString or ST_CircularString value.
ST_SRID	Returns the spatial reference identifier for the ST_Geometry as defined in spatial_ref_sys table.
ST_BOUNDARY	Returns the closure of the combinatorial boundary of this Geometry.

6.5 AGGREGATE FUNCTIONS

Aggregate functions operate on a set of geometry objects.

6.5.1 Aggregate Functions in Oracle Spatial

Table 6.9 illustrates the aggregate functions in Oracle Spatial [1].

6.5.2 PostGIS Aggregate Functions

Table 6.10 illustrates the aggregate functions in PostGIS [10].

Table 6.9. Aggregate Functions in Oracle Spatial

Function	Description
SDO_AGGR_MBR	Returns the minimum bounding rectangle of specified geometries
SDO_AGGR_UNION	Returns the union of geometry objects.
SDO_AGGR_CENTROID	Returns the centroid for the specified geometry objects.
SDO_AGGR_CONVEXHULL	Returns the convex hull for the specified geometry objects.

Table 6.10. PostGIS Aggregate Functions

Function	Description
ST_Extent	Similar to SDO_AGGR_MBR in Oracle.
ST_Accum	Constructs an array of geometries.
ST_Collect	Return a specified ST_Geometry value from a collection of other geometries.
ST_MakeLine	Creates a Linestring from point geometries.
ST_MemUnion	Same as ST_UNION, only memory-friendly.

CHAPTER 7

NETWORK MODELING

This chapter discusses the network model. With logical network information, you can analyze a network and answer questions, many of them related to path computing and tracing. In additional to logical network information, spatial information such as node locations and link geometries can be associated with the network.

7.1 ORACLE NETWORK MODELING

The network model contains logical information such as connectivity relationships among nodes and links, directions of links, and costs of nodes and links. With logical network information, you can analyze a network and answer questions, many of them related to path computing and tracing. In additional to logical network information, spatial information such as node locations and link geometries can be associated with the network. This information can help you to model the logical information (such as the cost of a route, because it's physical length can be directly computed from its spatial representation).

7.1.1 Network Modeling Concepts

The following are some key terms related to the network data model [13]:

- A node represents an object of interest.
- A link represents a relationship between two nodes. A link may be directed(that is, have a direction) or undirected (that is, not have a direction).
- A path is an alternating sequence of nodes and links, beginning and ending with nodes, and usually with no nodes and links appearing more than once.
- A network is a set of nodes and links. A network is directed if the links that is contains are directed, and a network is undirected if the links that it contains are undirected.
- A logical network contains connectivity information but no geometric information. This is the model used for network analysis. A logical network can
- be treated as a directed graph or undirected graph, depending on the application.

- A spatial network contains both connectivity information and geometric information. In a spatial network, the nodes and links are SDO_GEOMETRY geometry objects.
- Cost is a non-negative numeric attribute that can be associated with links or nodes for computing the minimum cost path, which is the path that has the minimum total cost from a start node to an end node.
- A spanning tree of a connected graph is a tree (that is, a graph with no cycles) that connects all nodes of the graph. (The directions of links are ignored in a spanning tree.) The minimum cost spanning tree is the spanning tree that connects all nodes and has the minimum total cost.

7.1.2 DATA Structures: The Network Tables

A network is defined using two tables: a node table and a link table. A network can also have a path table and a path link table. These tables are optional and are filled with the results of analyses performed, such as the shortest path between two nodes. They are needed only if applications want to make analysis results available to other applications by storing them inside the database. Refer to [13] for details on structure of each of the network tables. Figure 7.1 shows the relationship between the tables that describe a network.

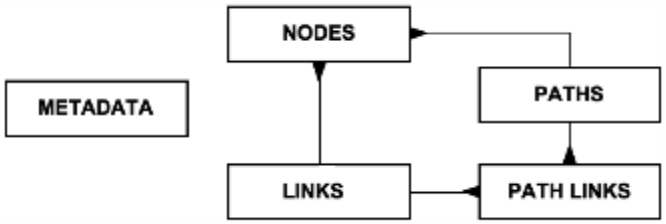

Figure 7.1. Network tables.

7.1.3 Network Data Model Application Programming Interface

The Oracle Spatial network data model includes two client application programming interfaces (APIs): a PL/SQL interface provided by the SDO_NET package and a Java interface [14]. Both interfaces let you create and update network data, and the Java interface provides network analysis capabilities.

Java API is used to perform network analysis operations such as the following:

- Shortest path (for directed and undirected networks): typical transitive closure problems in graph theory. Given a start and an end node, find the shortest path.

- Minimum cost spanning tree (for undirected networks): Given an undirected graph, find the minimum cost tree that connects all nodes.
- Reachability: Given a node, find all nodes that can reach that node, or find all nodes that can be reached by that node.
- Within-cost analysis (for directed and undirected networks): Given a target node and a cost, find all nodes that can be reached by the target node within the given cost.
- Nearest-neighbors analysis (for directed and undirected networks): Given a target node and number of neighbors, find the neighbor nodes and their costs to go to the given target node.
- All paths between two nodes: Given two nodes, find all possible paths between them.
- "Traveling salesman problem" analysis: Given a set of nodes, find the lowest-cost path that visits all nodes and in which the start and end nodes are the same.

7.1.4 Network Editor

Oracle provides a graphical editor that enables users to visualize the networks created and perform all the network analysis functions. Refer to [13] for details on how to run the network editor. Figure 7.2 shows a simple spatial network as viewed in Network Editor. Network editor provides various options to perform the network analysis functions. Figure 7.3 shows network analysis being performed on the spatial network through network editor.

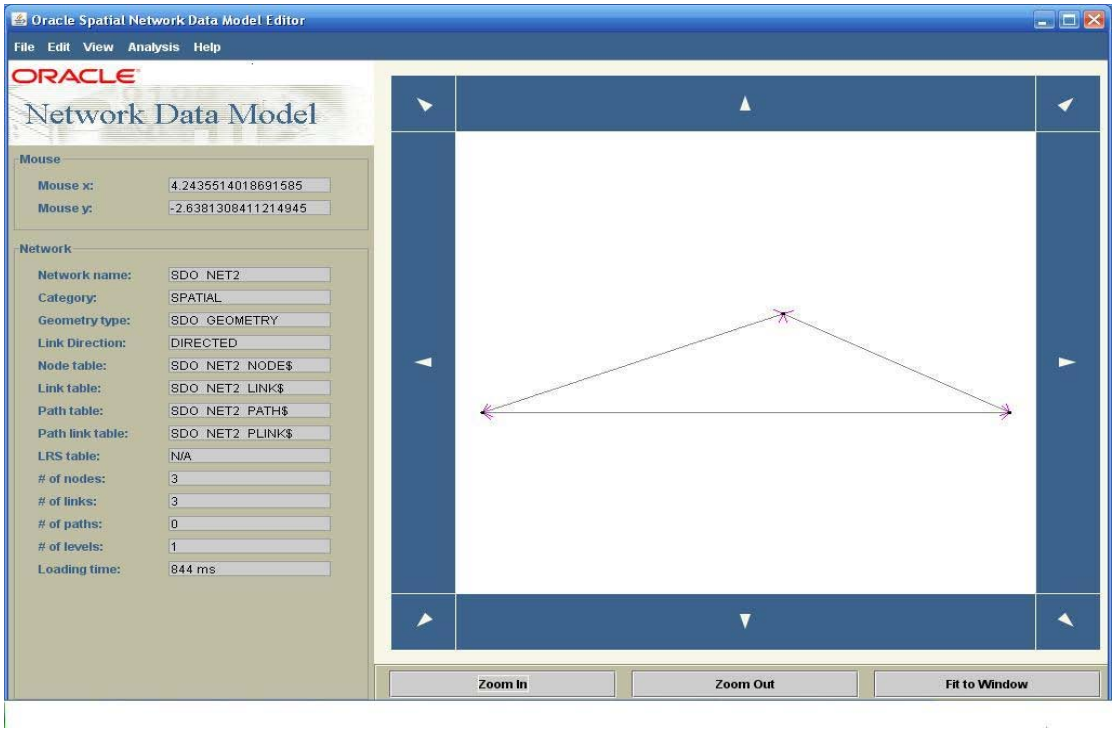

Figure 7.2. Simple spatial network.

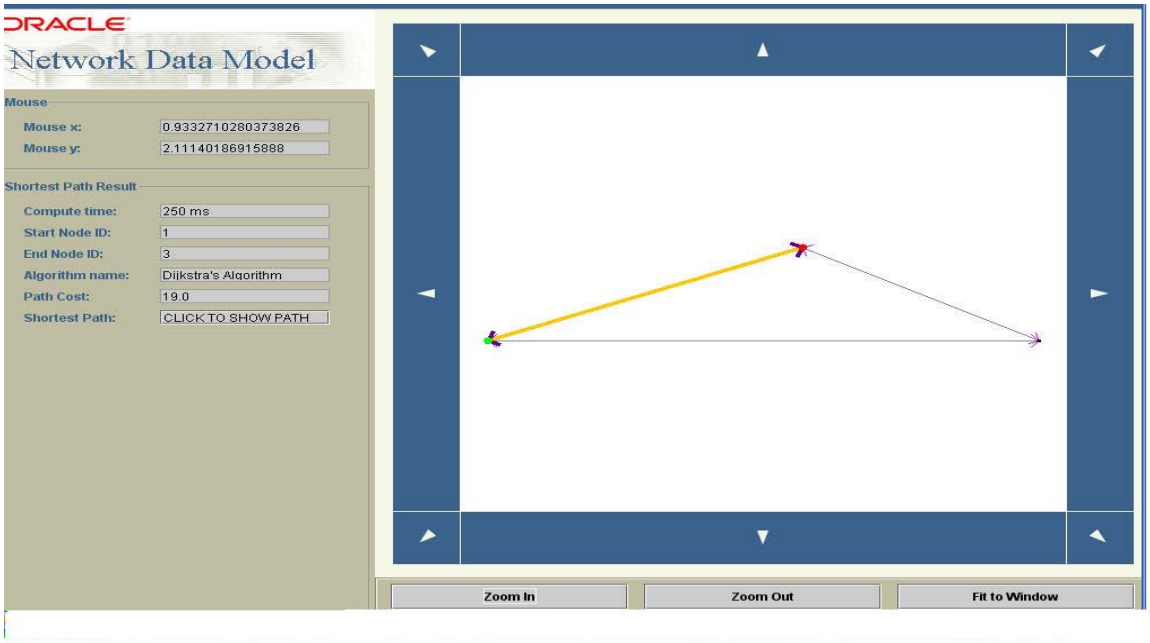

Figure 7.3. Shortest path from node 1 to node 3.

7.2 NETWORK MODELING IN POSTGIS

Network data model has still not been included as part of PostGIS. However, PostGIS provides functionality to solve the network routing problems using pgRouting [15]. pgRouting extends the PostGIS / PostgreSQL geospatial database to provide geospatial routing functionality. Refer to [11] to get details about installing pgRouting.

A few extra columns need to be added to the existing database tables to store input parameters and the solution. Then, the functions packaged with pgRouting can be executed to solve the various routing problems. Figure 7.4 shows the shortest route through the twin cities in United States calculated using pgRouting [11].

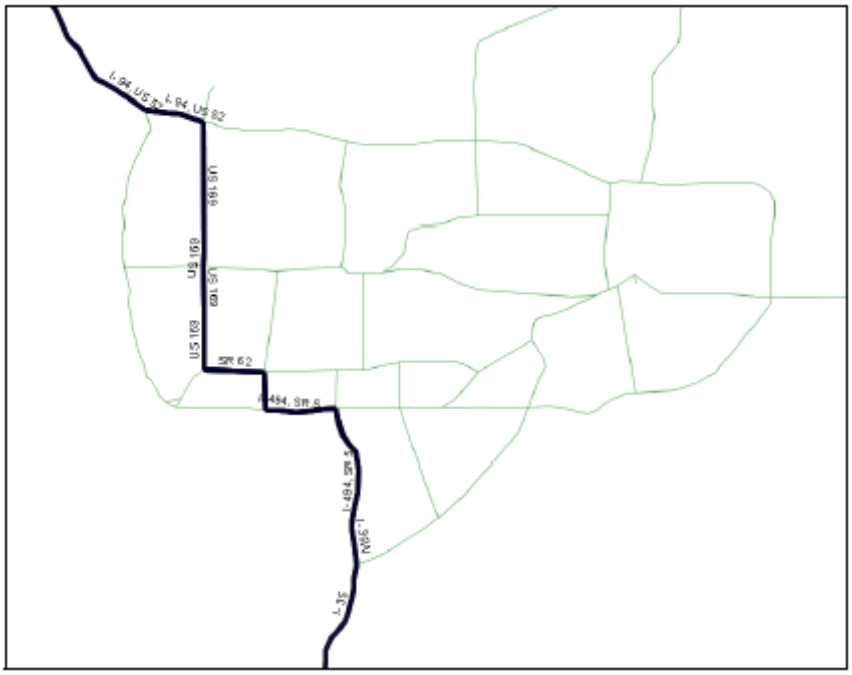

Figure 7.4. Shortest route through the Twin Cities.

CHAPTER 8

LOADING AND DISPENSING SPATIAL DATA

Several GIS vendors have their own formats to store spatial data. The ESRI shapefile format is one such example [16]. Spatial databases do not understand these formats. However, there are tools available to convert these shape files into database files, which can then be loaded into the database.

8.1 LOADING AND DISPENSING SPATIAL DATA IN ORACLE

This section provides details on loading and dispensing data in Oracle Spatial.

8.1.1 Loading Shape Files

Oracle spatial provides a command line utility, SHP2SDO that reads the shape files and outputs the SQL*Loader control and data files [17]. These files can then be used to populate the SDO_GEOMETRY column in an Oracle table. For more details on how to use the SHP2SDO utility refer [13].

8.1.2 Dispensing Spatial Data to Shape Files

Oracle does not provide any support for dispensing spatial data.

8.2 LOADING AND DISPENSING SPATIAL DATA IN POSTGIS

This section provides details on loading and dispensing spatial data in PostGIS.

8.2.1 Loading Shape Files in PostGIS

PostGIS offers an easy way to load shape files in the database. It provides a shp2pgsql-gui which imports shape files and converts them into tables. For details on using the shp2pgsql-gui refer [11]. Figure 8.1 shows using sh2pgsql-gui to load the bc_pubs table.

8.2.2 Dispensing Spatial Data to Shape Files

PostGIS allows us to create shape files out of the database tables using the pgsql2shp tool that comes packaged with PostGIS [11]. This is a command line tool to output PostGIS

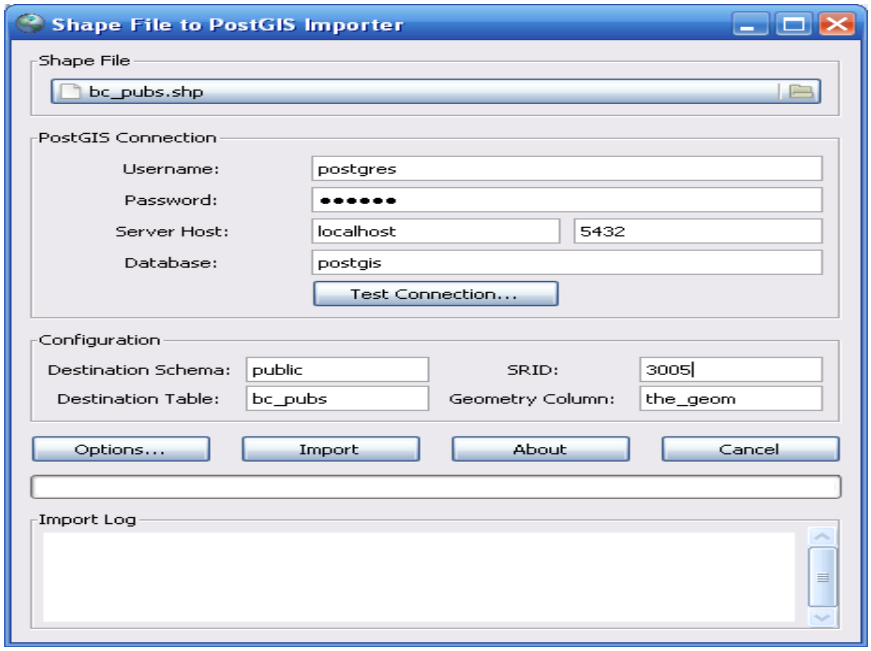

Figure 8.1. Using shp2pgsql-gui to load the bc_pubs table.

spatial data to ESRI shapefile format. It outputs .shp, .shx, .dbf, .prj files. The .prj file is output only if the projection is known, for example, if you didn't use an SRID of -1(which stands for null srid). The following command shows the creation of shape file for bc_pubs table loaded under the postgis database:

C:\Program Files\PostgreSQL\8.4\bin>pgsql2shp -u postgres -P potato postgis bc_pubs

Initializing... Done (postgis major version: 1).

Output shape: Point

Dumping: XXXXXX [417 rows].

CHAPTER 9

VIEWING SPATIAL DATA

Spatial data loaded under databases can be visualized using various tools.

9.1 ORACLE MAPVIEWER

Oracle MapViewer is a programmable tool for rendering maps using spatial data managed by Oracle Spatial. The latest version of MapViewer can be installed from [18]. The absence of a utility to go from SDO to shapefiles means the user has to confront the rather fussy Map Viewer. Figure 9.1 shows the cola_markets table as viewed in MapViewer.

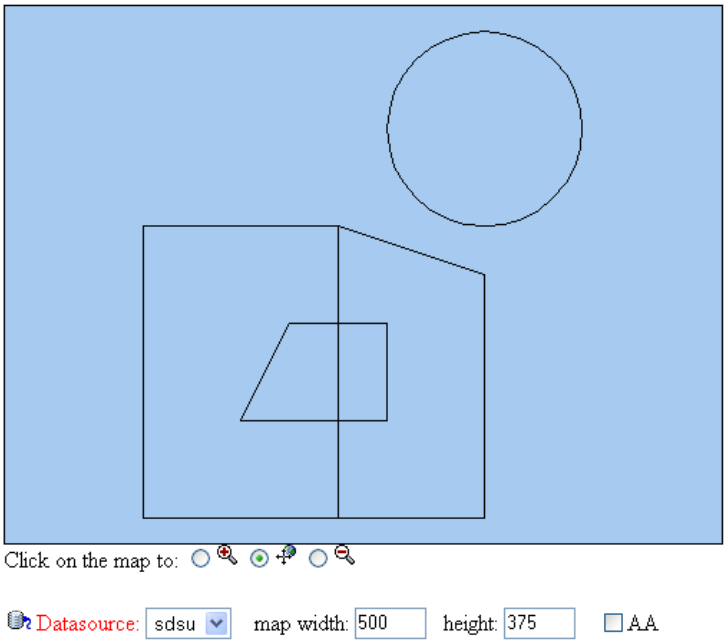

Figure 9.1. Map viewer showing the cola_markets table.

9.2 VIEWING SPATIAL DATA IN POSTGIS

Desktop viewers for PostGIS data cannot render curved geometries. There are a number of open source options for desktop viewers / editors of PostGIS data:

- QGIS, a C++ / Qt program;

- uDig, a Java / Eclipse program; and,
- gvSIG, a Java / Swing program

Quantum GIS (QGIS) has been used in this thesis to render spatial data. QGIS is available for download at [19]. Figure 9.2 shows the cola_markets table as viewed in QGIS.

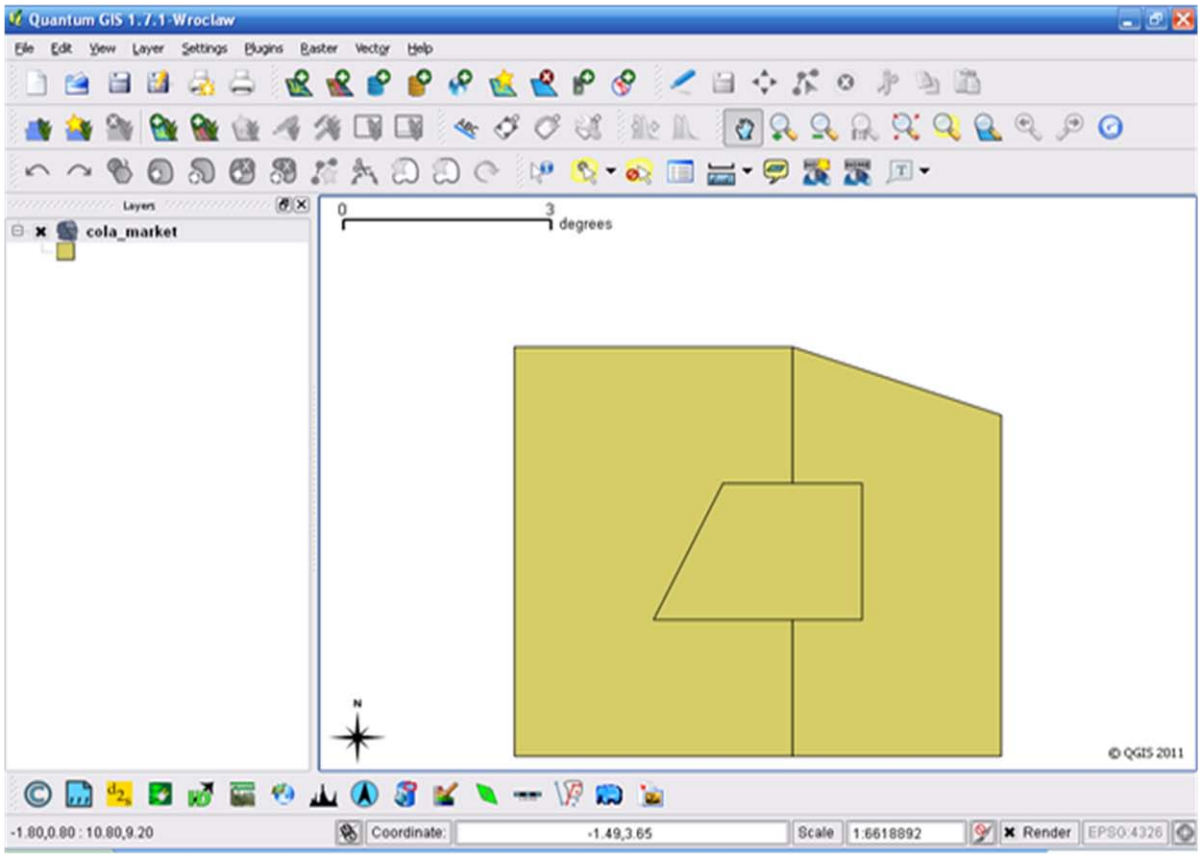

Figure 9.2. QGIS showing the cola_market table.

CHAPTER 10

CONCLUSION

Oracle Spatial and PostGIS are the most mature implementations of a spatial type system and are known for their relevant host databases. However, the research done as a part of this thesis suggests that PostGIS is more advantageous over Oracle Spatial for the reasons stated below:

1. Cost Savings - PostGIS is an open source spatial database; hence is virtually free. On the other hand, Oracle is a commercial database; hence software licensing for the server is several times costlier than the hardware on which it runs. This makes it unsuitable for smaller projects.
2. Ease of Integration - Majority products support PostGIS as a data source. Below lists a small subset of products that are compatible with PostGIS:
    a. Mapserver
    b. Geotools (Geoserver, uDig)
    c. FDO (Mapguide, Autodesk Map 3D)
    d. JUMP (OpenJUMP, Kosmo)
    e. OGR (QGIS, Mapserver, GRASS)
    f. FME (ArcGIS Data Interoperability Extension)
    g. Cadcorp SIS
    h. Manifold
    i. ESRI ArcSDE 9.3
    j. Python / Perl / PHP
3. Ease of use - Creation of spatial objects is simpler in PostGIS. For e.g., PostGIS polygon can be created as follows:
   POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))
   Similar polygon in Oracle Spatial can be created as:
   MDSYS.SDO_GEOMETRY(
   2003, NULL, NULL, MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,1),
   MDSYS.SDO_ORDINATE_ARRAY(0,0, 0,1, 1,1, 1,0, 0,0))
4. Superior performance - PostGIS uses “lightweight” implementations by using minimum amount of bytes to express the geometries and indexes. This essentially minimizes the number of write operations to the disk which in turn improves the overall performance.

5. Feature rich - PostGIS offers numerous extra functions that are not available in Oracle Spatial. For example, PostGIS offers more output formats than Oracle Spatial using functions like ST_AsKML, ST_AsGML, ST_STARTPOINT, and so on. For other functions refer to [11].
6. Lower disk usage - Postgres requires approximately 532 MB of disk space, whereas Oracle requires approximately 3.3 GB.

On the other hand, Oracle Spatial does have some benefits such as good tech support, strong developer community, and training programs, which PostGIS lacks. This explains why many "risk averse" companies are flocking toward Oracle Spatial.

CHAPTER 11

FUTURE WORK

1. Compare open source databases MySQL Spatial and Postgres Spatial.
2. Compare Oracle Spatial, SQL Server and DB2.
3. Detailed study of migration from Oracle Spatial to Postgres Spatial.

REFERENCES

[1] C. MURRAY, *Oracle spatial developer's guide 11g release 1 (11.1)*, Oracle, Redwood Shores, California, 2006.

[2] ORACLE, *Oracle database software downloads*, Oracle, http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html/, accessed September 2011, n.d.

[3] POSTGIS, *Downloads*. PostGIS, http://www.postgis.org/download/, accessed September 2011, n.d.

[4] OGC, *Geospatial and location standards*. OGC, http://www.opengis.org/, accessed December 2011, n.d.

[5] WIKIPEDIA, *Well-known text*. Wikipedia, http://en.wikipedia.org/wiki/Well-known_text, accessed October 2011, 2011.

[6] POSTGIS, *Using PostGIS: Data management and queries*, PostGIS, http://postgis.org/docs/ch04.html#Create_Spatial_Table, accessed November 2011, n.d.

[7] C. MURRAY, *Oracle spatial user's guide and reference release 9.0.1*, Oracle, Redwood Shores, California, 2001.

[8] A. VELICANU AND S. OLARU, *Optimizing spatial databases*, Informatica Economica 14 (2010), pp. 61-71.

[9] M. SARDADI, M. RAHIM, Z. JUPRI, AND D. DAMAN. *Choosing R-tree or quadtree spatial data indexing in one Oracle spatial database system to make faster showing geographical map in mobile geographical information system technology*, World Academy of Science Engineering and Technology 46 (2008), pp. 249-257.

[10] POSTGIS, *1.4.2 manual*. POSTGIS, http://postgis.refractions.net/documentation/manual-1.4/, accessed December 2011, n.d.

[11] R. OBE AND L. HSU, *PostGIS in action*, Manning Publications Co., Greenwich, Connecticut, 2011.

[12] WIKIPEDIA, *DE-9IM*. Wikipedia, http://en.wikipedia.org/wiki/DE-9IM, accessed, November 2011, 2011.

[13] R. KOTHURI, A. GODFRIND AND E. BEINAT, *Pro Oracle spatial*, University of Michigan Press, Ann Arbor, Michigan, 2004.

[14] C. MURRAY, *Topology and network data models 10g release 1 (10.1)*, Oracle, Redwood Shores, California, 2003.

[15] PGROUTING, *Homepage*. pgRouting, http://pgrouting.org/, accessed December 2011, n.d.

[16] WIKIPEDIA, *Esri*. Wikipedia, http://en.wikipedia.org/wiki/Esri/, accessed October 2011, n.d.

[17] ORACLE UTILITIES, *SQL loader*, Oracle Utilities, http://www.oracleutilities.com/OSUtil/sqlldr.html/, accessed September 2011, n.d.

[18] ORACLE, *Mapviewer downloads*. Oracle, http://www.oracle.com/technetwork/java/javase/downloads/index.html, accessed October 2011, n.d.

[19] QUANTUM GIS, *Homepage*. Quantum GIS, http://www.qgis.org/, accessed November 2011, n.d.

APPENDIX

POSTGIS USER MANUAL

The below manual shall provide details to operate PostGIS database.

Section 1: Database Installation

This section provides details on database installation.

1.1 Postgres Installation

Download the latest version of PostgreSQL available from http://www.postgresql.org/download/windows/ . This is a one click installer that includes the PostgreSQL server, pgAdmin III; a graphical tool for managing and developing your databases, and StackBuilder ; a package manager that can be used to download and install additional PostgreSQL applications and drivers.
Note: - Remember the password entered during installation. The user name defaults to postgres.

Step I: Welcome page appears as shown in Figure A.1.

Figure A.1. Welcome page.

Step 2: Select the Installation Directory as shown in Figure A.2.

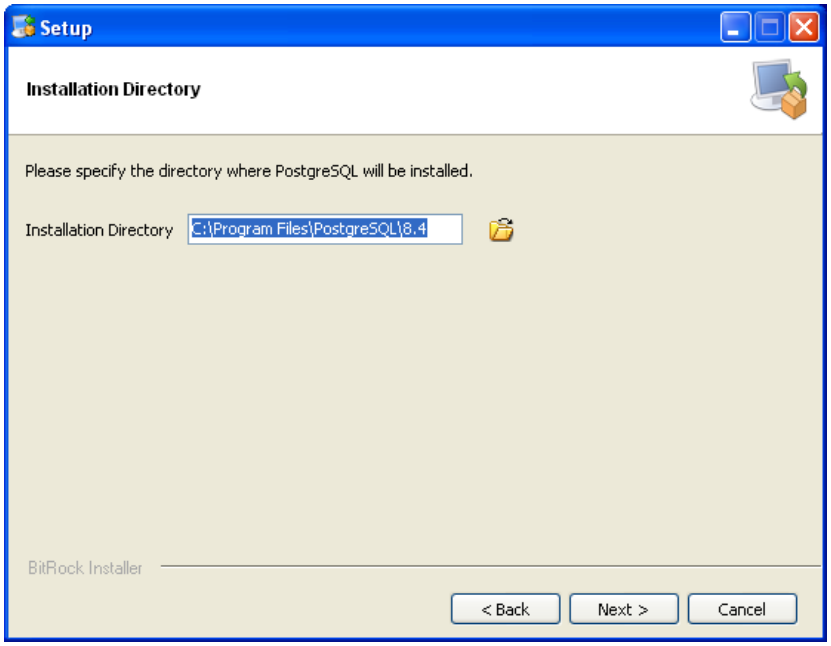

Figure A.2. Installation directory.

Step 3: Select the Data Directory as shown in Figure A.3.

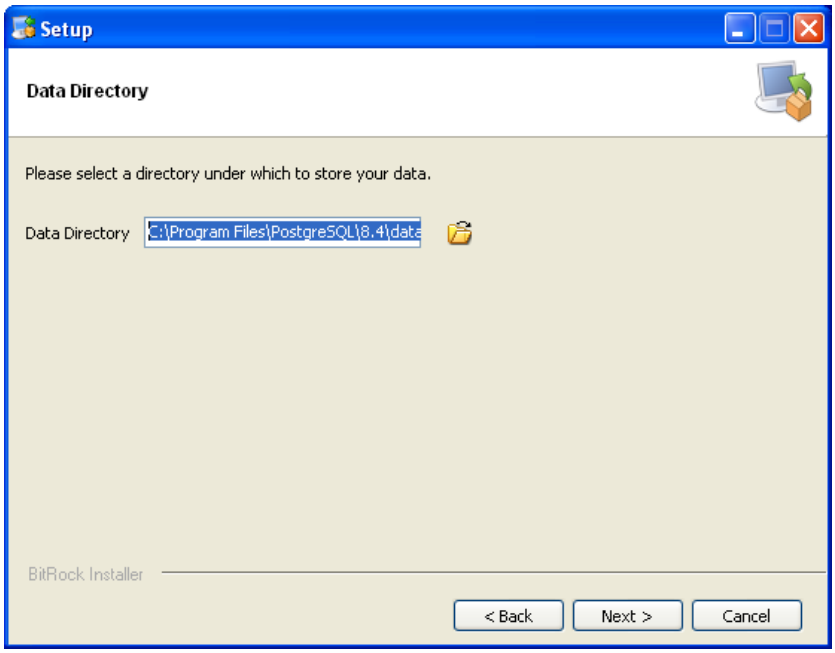

Figure A.3. Installation directory.

Step 4: Enter a password for the newly-created "postgres" superuser account as shown in Figure A.4. Remember this password!

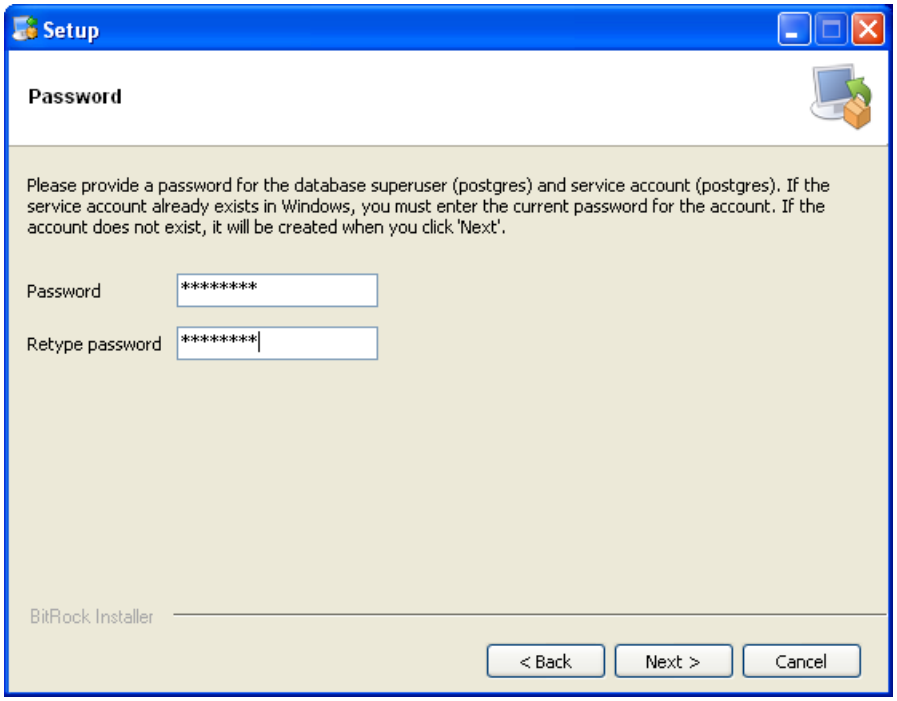
Setup
Password
Please provide a password for the database superuser (postgres) and service account (postgres). If the service account already exists in Windows, you must enter the current password for the account. If the account does not exist, it will be created when you click 'Next'.
Password
****
Retype password
****
BitRock Installer
< Back
Next >
Cancel

Figure A.4. Password.

Step 5: Enter Port number as shown in Figure A.5.

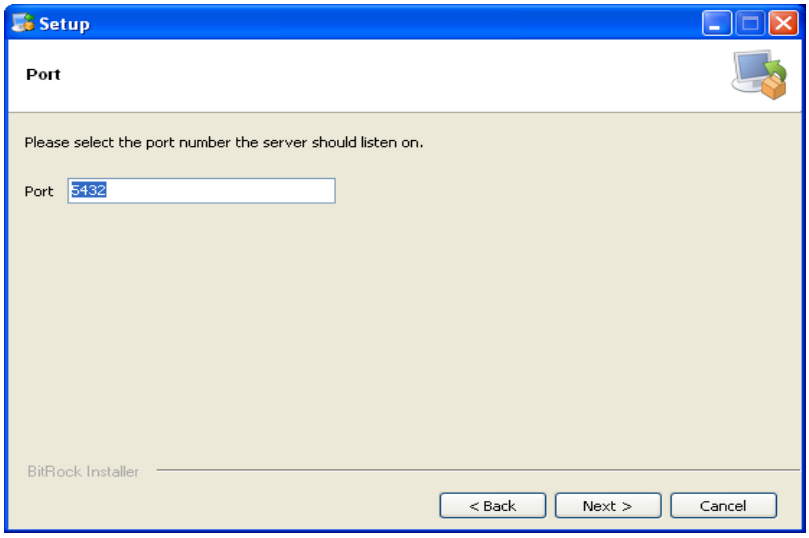

Figure A.5. Port number.

Step 6: Select the [Default locale] option. The Default Locale option reflects the locale (location) setting of the host operating system. Uncheck the option that says Install pl/pgsql in template1 database. Click Next as shown in Figure A.6.

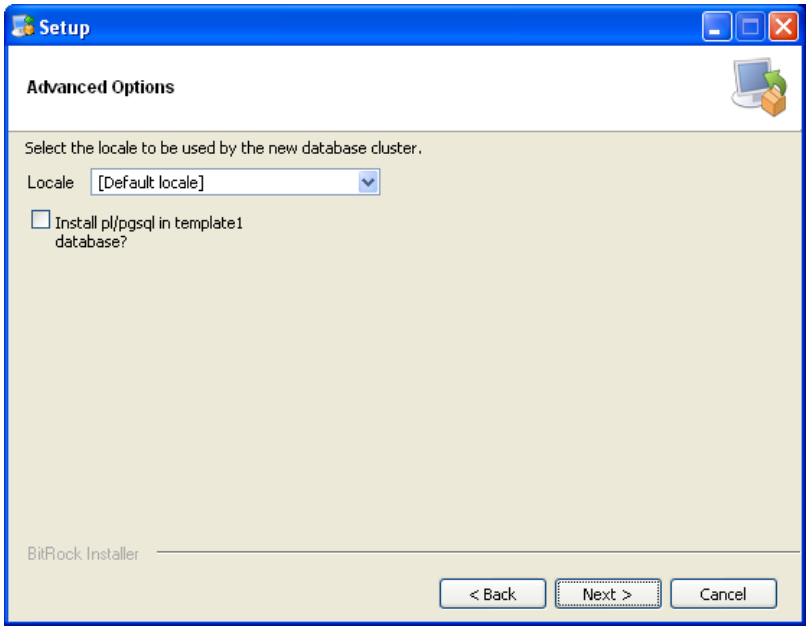

Figure A.6. Advanced options.

Step 7: Ready to Install. Click next as shown in Figure A.7.

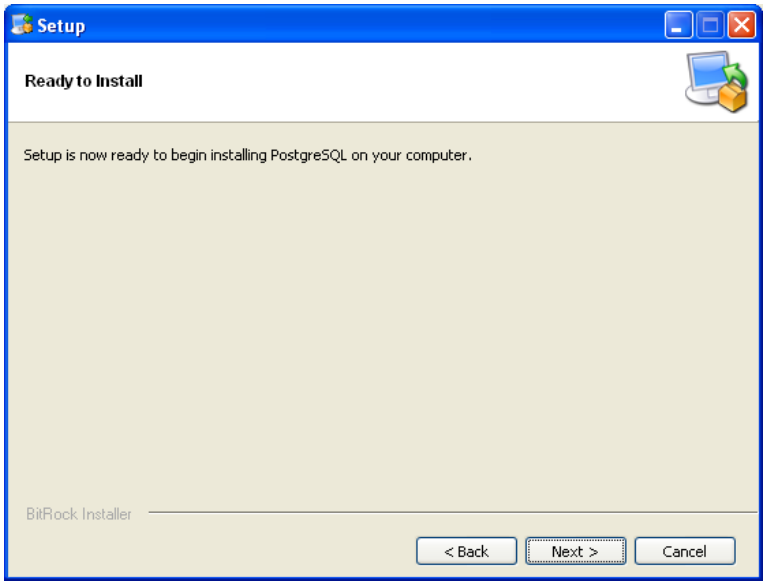

Figure A.7. Ready to install.

Step 8: It will perform installation as shown in Figure A.8.

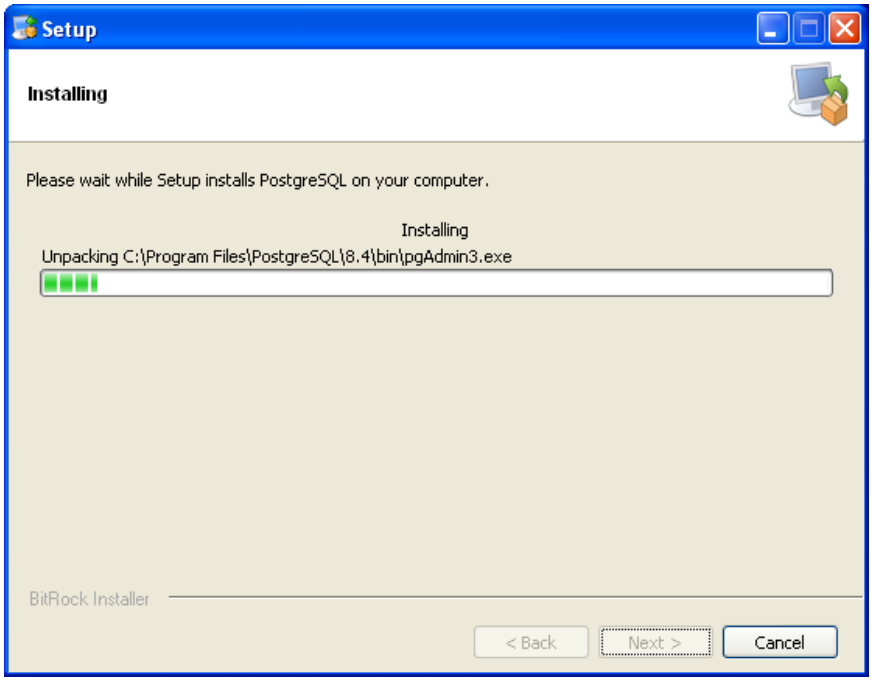

Figure A.8. Installing.

Step 9: Ensure that the Launch Stack Builder at exit? option is not selected, then click Finish as shown in Figure A.9.

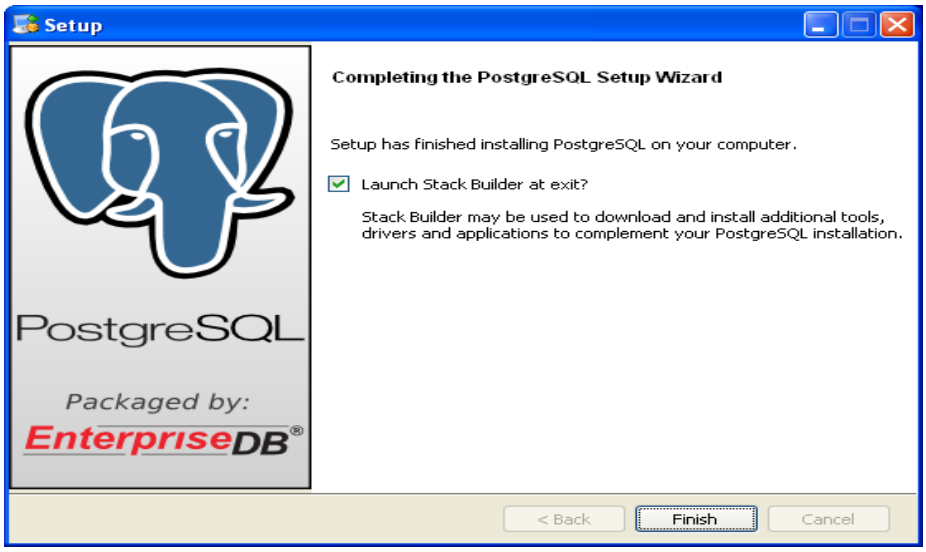
Setup
PostgreSQL
Packaged by:
EnterpriseDB
Completing the PostgreSQL Setup Wizard
Setup has finished installing PostgreSQL on your computer.
Launch Stack Builder at exit?
Stack Builder may be used to download and install additional tools, drivers and applications to complement your PostgreSQL installation.
< Back
Finish
Cancel

Figure A.9. Setup complete.

1.2 PostGIS Installation

The next step is to install the PostGIS add-in to create spatially enabled databases.

Step 1: PostGIS is included as part of the new PostgreSQL StackBuilder architecture. The latest version can be installed by going to Start -> Programs -> PostgreSQL 8.4 -> Application Stack Builder. Select your PostgreSQL installation from the list and then click "Next" as shown in Figure A.10.

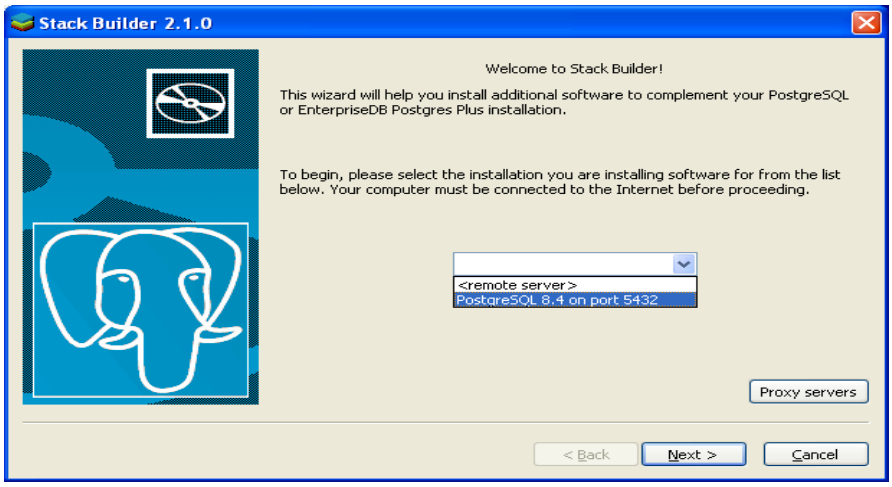
Stack Builder 2.1.0
Welcome to Stack Builder!
This wizard will help you install additional software to complement your PostgreSQL or EnterpriseDB Postgres Plus installation.
To begin, please select the installation you are installing software for from the list below. Your computer must be connected to the Internet before proceeding.
<remote server>
PostgreSQL 8.4 on port 5432
Proxy servers
< Back
Next >
Cancel

Figure A.10. Welcome to stack builder.

Step 2: There are a number of categories to choose from. PostGIS is available under the "Spatial Extensions" group. Select the latest version and click Next as shown in Figure A.11.

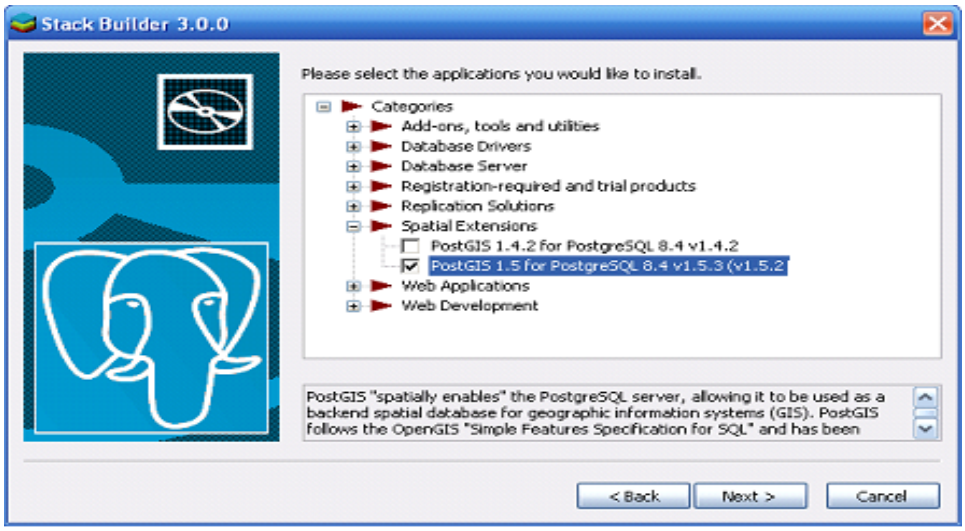
Stack Builder 3.0.0
Please select the applications you would like to install.
Categories
Add-ons, tools and utilities
Database Drivers
Database Server
Registration-required and trial products
Replication Solutions
Spatial Extensions
PostGIS 1.4.2 for PostgreSQL 8.4 v1.4.2
PostGIS 1.5 for PostgreSQL 8.4 v1.5.3 (v1.5.2
Web Applications
Web Development
PostGIS "spatially enables" the PostgreSQL server, allowing it to be used as a backend spatial database for geographic information systems (GIS). PostGIS follows the OpenGIS "Simple Features Specification for SQL" and has been
< Back
Next >
Cancel

Figure A.11. Select category.

Step 3: There is only one Australian mirror for PostgreSQL downloads. Select the FTP option and click next as shown in Figure A.12.

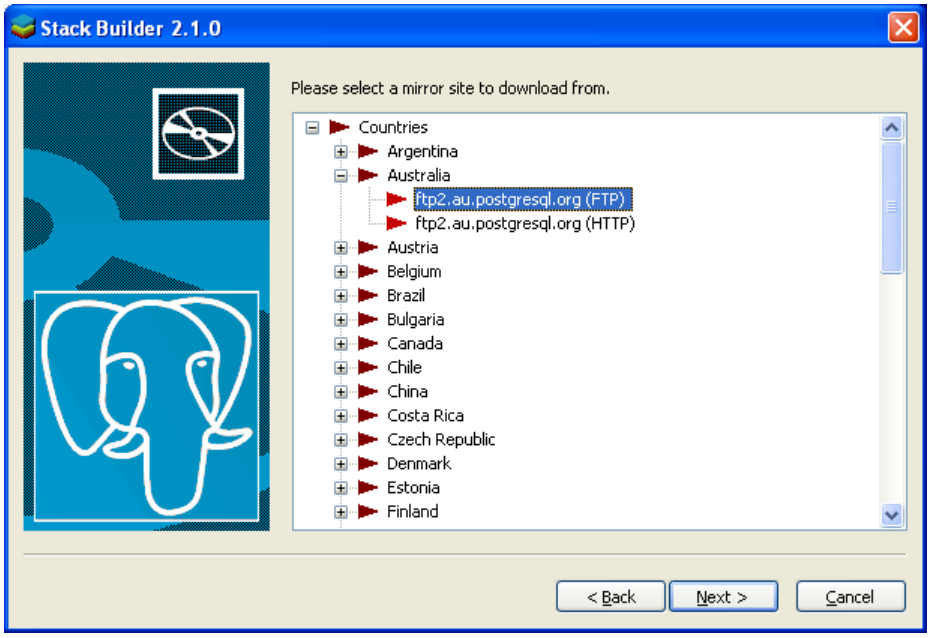

Figure A.12. Select mirror site.

Step 4: The review page lists all selected packages. Click Next as shown in Figure A.13.

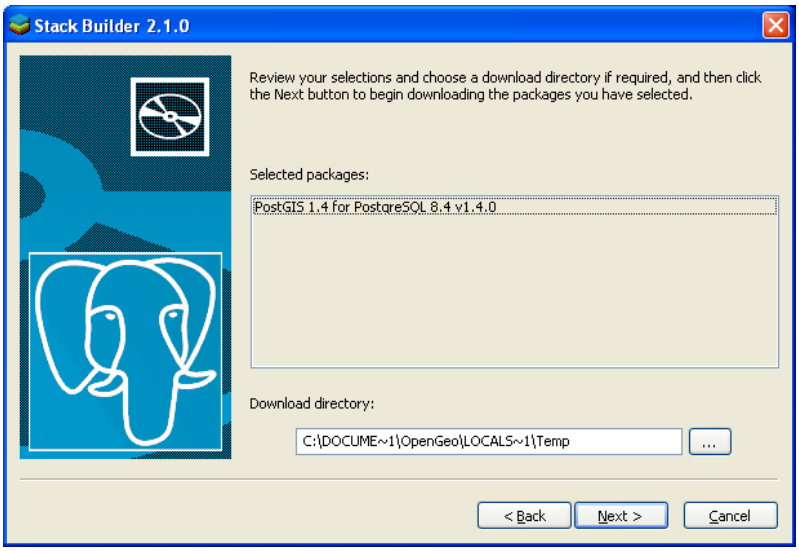
Stack Builder 2.1.0
Review your selections and choose a download directory if required, and then click the Next button to begin downloading the packages you have selected.
Selected packages:
PostGIS 1.4 for PostgreSQL 8.4 v1.4.0
Download directory:
C:\DOCUME~1\OpenGeo\LOCALS~1\Temp
...
< Back
Next >
Cancel

Figure A.13. Selected packages.

Step 5: The download is small and shouldn't take very long as shown in Figure A.14

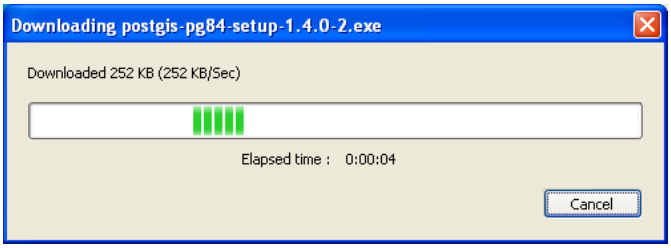

Figure A.14. Downloading.

Step 6: Once the download is complete you will be presented with the following screen. Click Next to continue as shown in Figure A.15.

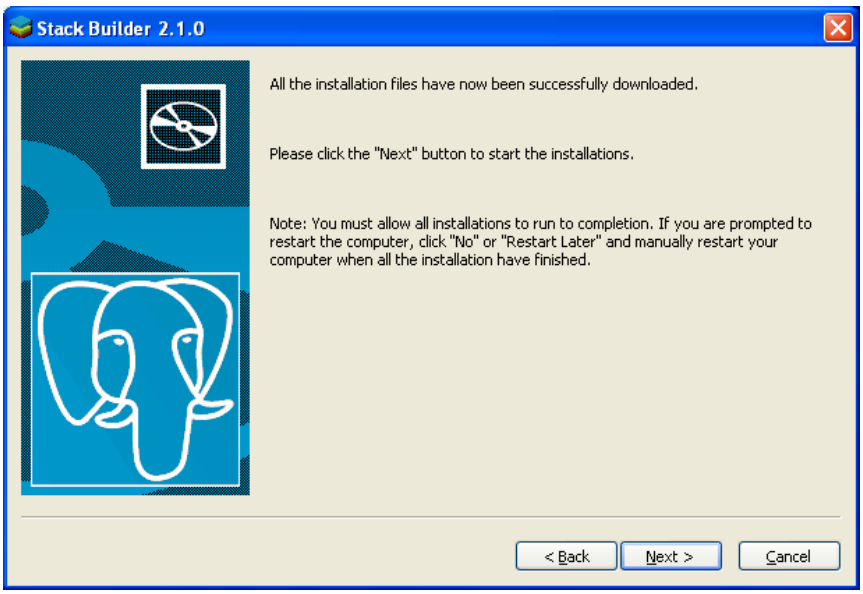
Stack Builder 2.1.0
All the installation files have now been successfully downloaded.
Please click the "Next" button to start the installations.
Note: You must allow all installations to run to completion. If you are prompted to restart the computer, click "No" or "Restart Later" and manually restart your computer when all the installation have finished.
< Back
Next >
Cancel

Figure A.15. Installation files downloaded.

Step 7: PostGIS is licensed under the GNU General Public License, the text of which is displayed in the following dialog. Select I Agree to continue as shown in Figure A.16.

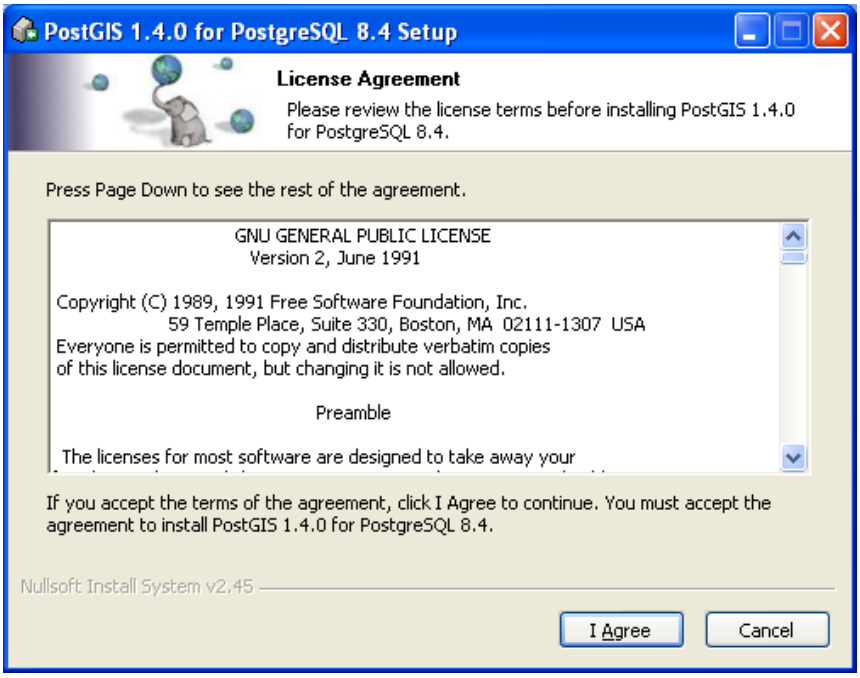
PostGIS 1.4.0 for PostgreSQL 8.4 Setup
License Agreement
Please review the license terms before installing PostGIS 1.4.0 for PostgreSQL 8.4.
Press Page Down to see the rest of the agreement.
GNU GENERAL PUBLIC LICENSE
Version 2, June 1991
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
Preamble
The licenses for most software are designed to take away your
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install PostGIS 1.4.0 for PostgreSQL 8.4.
Nullsoft Install System v2.45
I Agree
Cancel

Figure A.16. License agreement.

Step 8: The PostGIS installer can create a spatially enabled database automatically after installation. Uncheck the Create spatial database box and click Next as shown in Figure A.17.

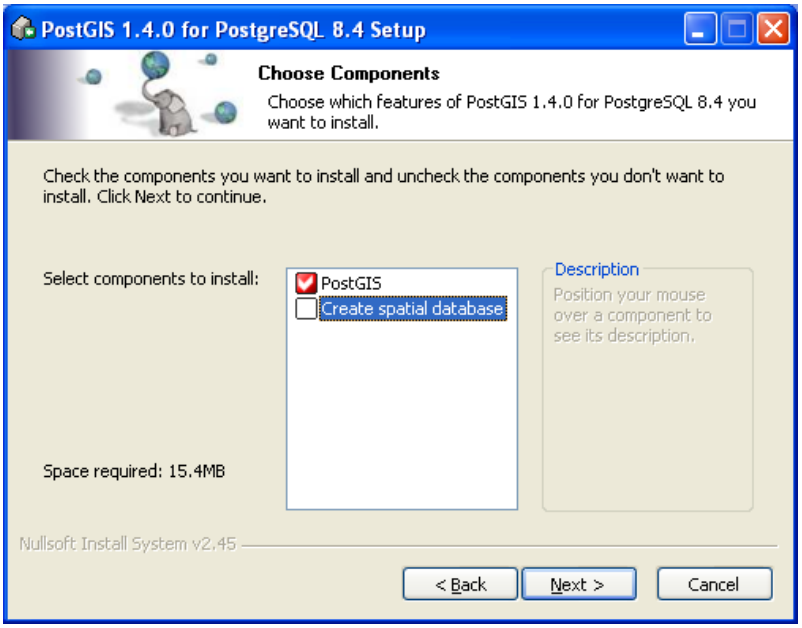
PostGIS 1.4.0 for PostgreSQL 8.4 Setup
Choose Components
Choose which features of PostGIS 1.4.0 for PostgreSQL 8.4 you want to install.
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.
Select components to install:
PostGIS
Create spatial database
Description
Position your mouse over a component to see its description.
Space required: 15.4MB
Nullsoft Install System v2.45
< Back
Next >
Cancel

Figure A.17. Choose components.

Step 9: PostGIS needs to know the location of the target PostgreSQL installation. It will determine this automatically if PostgreSQL is installed in the default location. Click Next to continue as shown in Figure A.18.

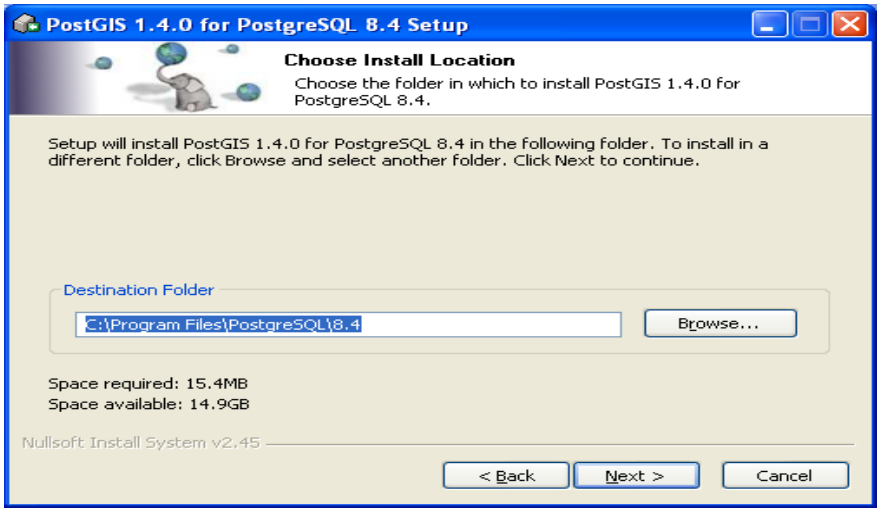
PostGIS 1.4.0 for PostgreSQL 8.4 Setup
Choose Install Location
Choose the folder in which to install PostGIS 1.4.0 for PostgreSQL 8.4.
Setup will install PostGIS 1.4.0 for PostgreSQL 8.4 in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.
Destination Folder
C:\Program Files\PostgreSQL\8.4
Browse...
Space required: 15.4MB
Space available: 14.9GB
Nullsoft Install System v2.45
< Back
Next >
Cancel

Figure A.18. Choose install location.

Step 10: Enter the password and the port number you defined for the “postgres” user when installing PostgreSQL and click Next to continue as shown in Figure A.19.

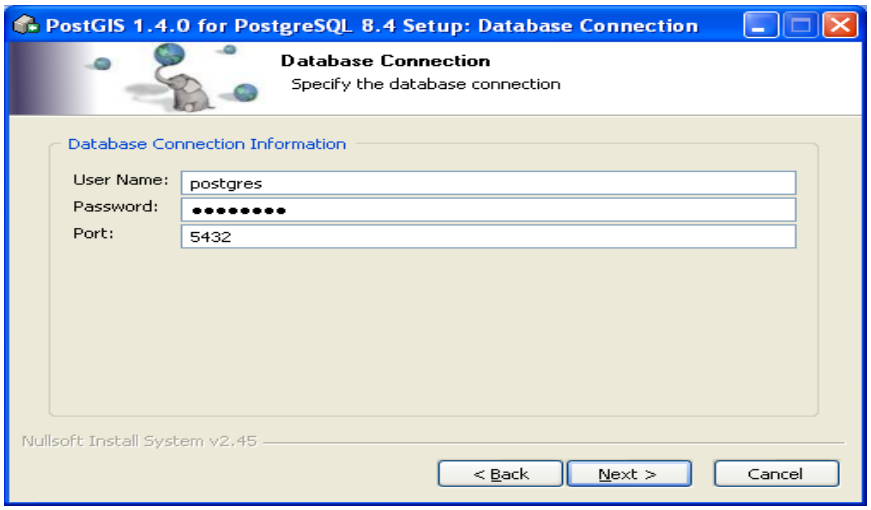

Figure A.19. Database connection.

Step 11: After installation, click Close to return to the Stack Builder as shown in Figure A.20.

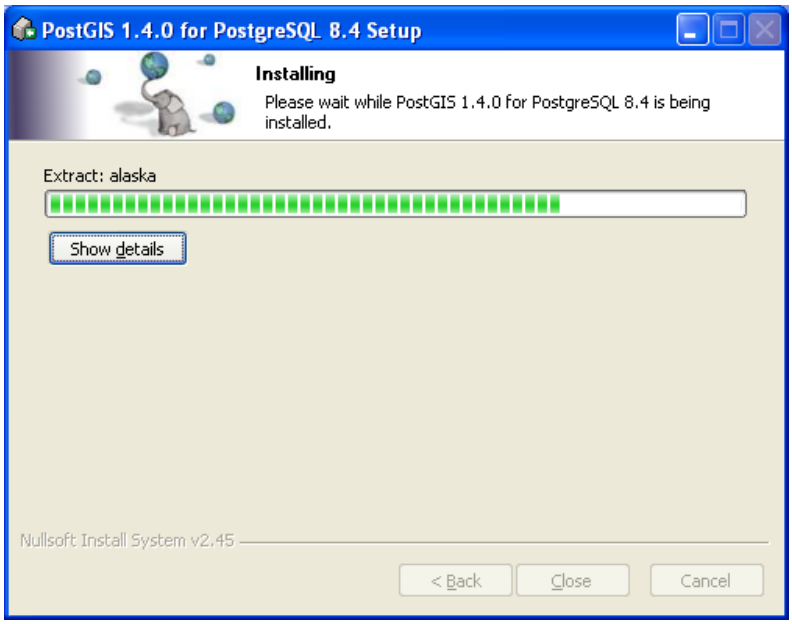

Figure A.20. Installing PostGIS.

Step 12: The installation is now complete. Click Finish to exit as shown in Figure A.21.

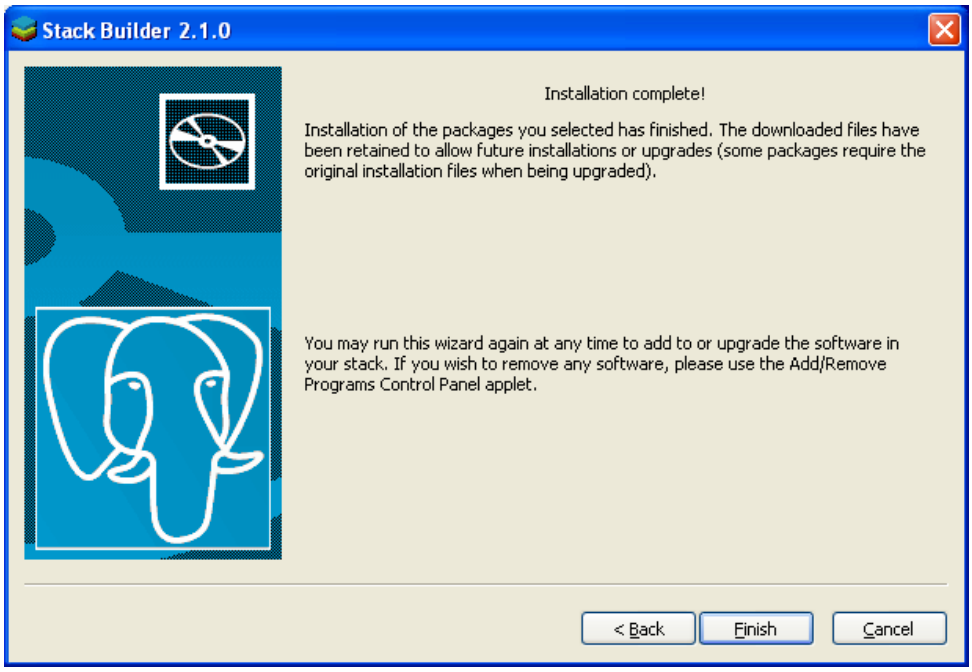
Stack Builder 2.1.0
Installation complete!
Installation of the packages you selected has finished. The downloaded files have been retained to allow future installations or upgrades (some packages require the original installation files when being upgraded).
You may run this wizard again at any time to add to or upgrade the software in your stack. If you wish to remove any software, please use the Add/Remove Programs Control Panel applet.
< Back
Finish
Cancel

Figure A.21. Installation complete.

1.3 Spatially-Enable PostgreSQL

Note: - The PostgreSql server needs to be started before connecting to the database.

The installer will add a PostgreSQL menu to your Start menu.

- Navigate to Start -> Programs -> PostgreSQL 8.4 and run “Start Server.”
- Once the server is up and running, navigate to Start -> Programs -> PostgreSQL 8.4 and run PgAdmin III (When PgAdminIII is closed, the PostgreSql server needs to be stopped by navigating to Start -> Programs -> PostgreSQL 8.4 and running “Stop Server.”)
- In PgAdmin III , double click on the “PostgreSQL Database Server” tree entry. You will be prompted for the super user password as shown in Figure A.22.

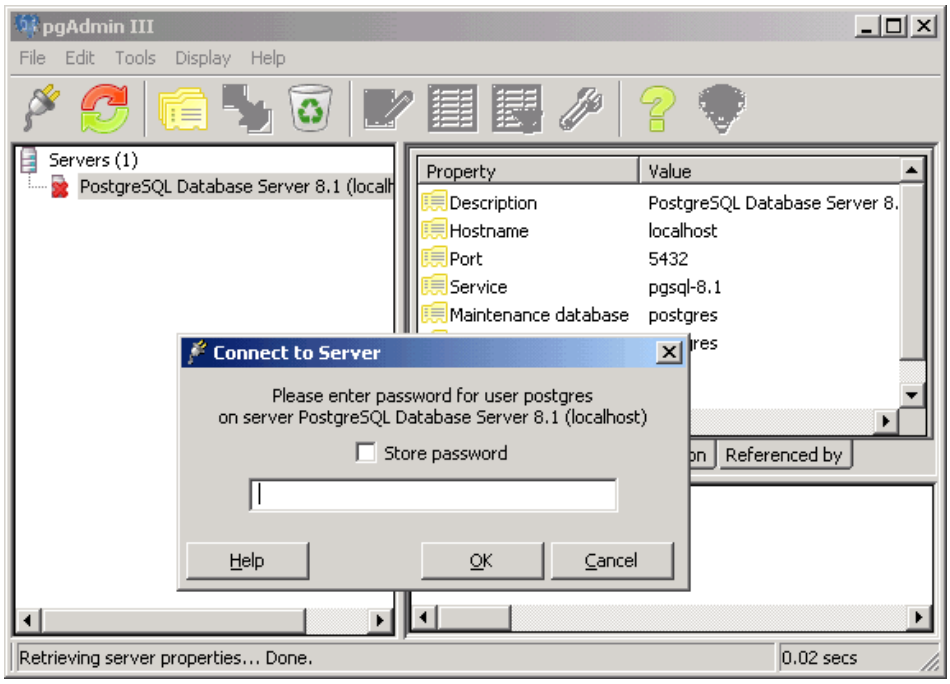

Figure A.22. pgAdmin III.

- Navigate to the “Databases” section of the database tree and open “Edit ⇨ New Object ⇨ New Database. Add a new database named “postgis”, with “postgres” as the owner “template_postgis” as the template. Click OK as shown in Figure A.23. By using the “template_postgis” database as the template, you get a new database with spatial capabilities already installed and enabled.

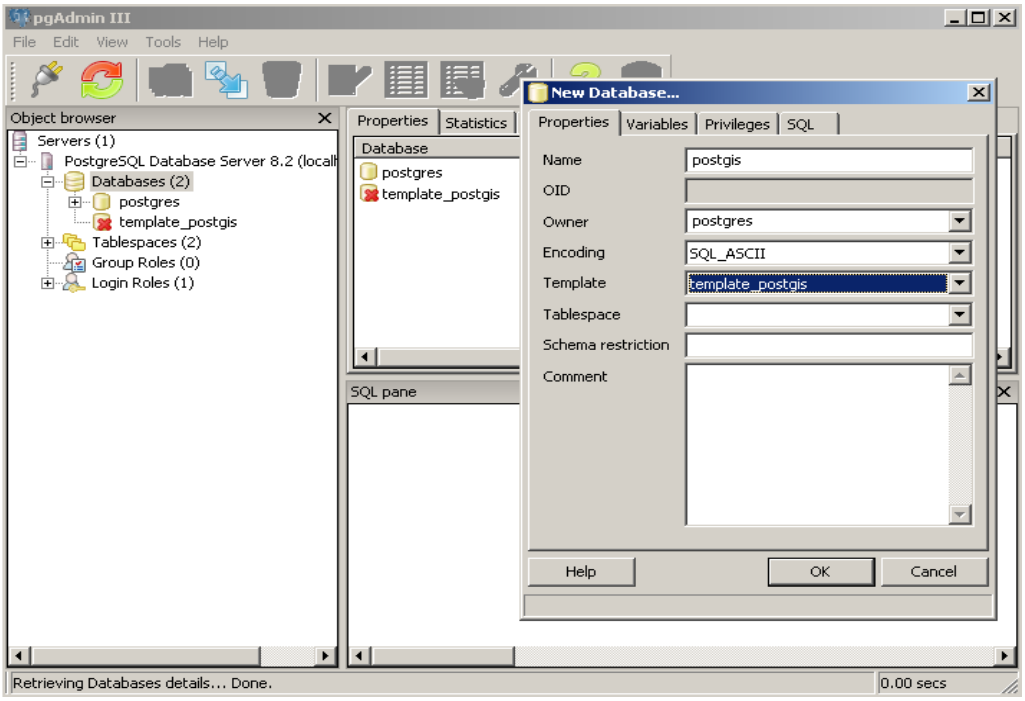

Figure A.23. New database creation.

Open the new “postgis” database. Navigate to postgis ⇨ Schemas ⇨ public ⇨ Tables and see what tables exist. You should see “geometry_columns” and “spatial_ref_sys” tables as shown in Figure A.24, which are standard tables created by PostGIS. The tables starting in “pg_ts_” are used by the full-text-search module and can be ignored. They may not appear in latest versions of PostgreSQL.

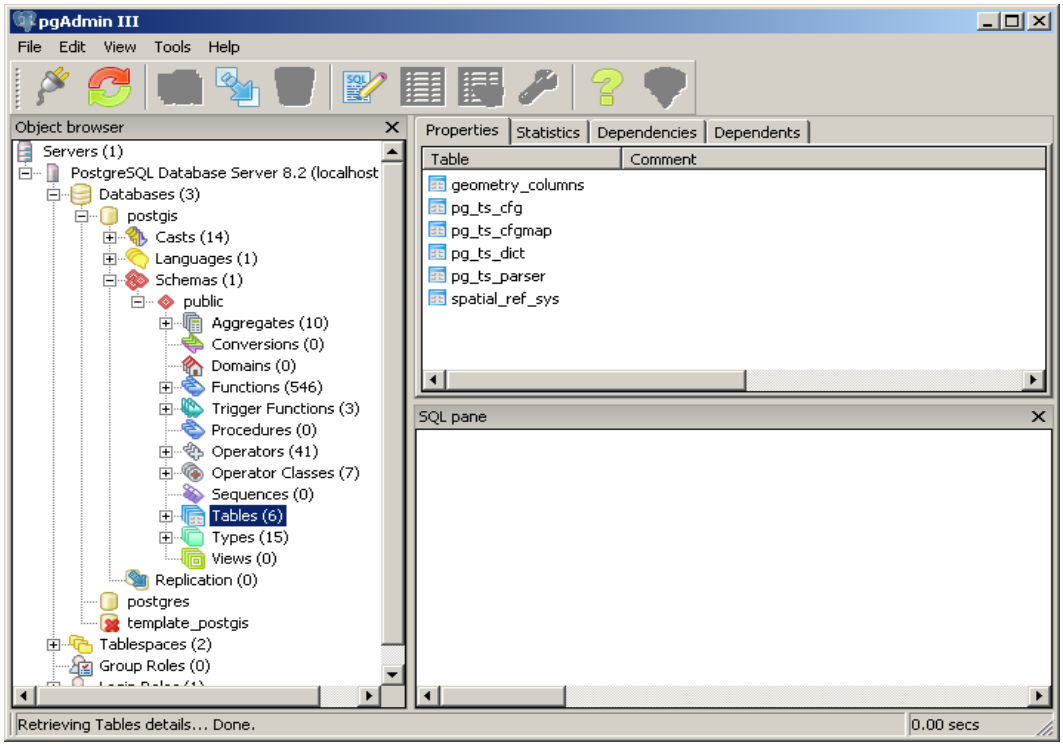

Figure A.24. Tables with PostGIS installed.

SECTION 2: USING POSTGIS

This section provides details on how to use PostGIS.

2.1 Simple Spatial SQL

This section tests the creation of a table with geometry column, adds some spatial objects to the table, and runs a few spatial functions against the table contents.

Once the "postgis" database gets created in pgAdmin III, select "postgis" (it should get highlighted), and open up the query tool window, using the Tools ⇨ Query Tool or the button with the "SQL" icon on it.

Paste the following SQL into the query tool window as shown in Figure A.25, and then hit the green triangle "Execute" button (or the F5 shortcut key).

```
--Create table
CREATE TABLE cola_market (
mkt_id int2,
name VARCHAR(32),geom GEOMETRY);
--Inserts geometries with SRID as -1(stands for null) into the table
```

```
INSERT INTO cola_market (mkt_id,name,geom)
VALUES (1,'cola_a', ST_GeomFromText('POLYGON((1 1,1 7,5 7,5 1,1 1))',-1));
INSERT INTO cola_market (mkt_id,name,geom)
VALUES (2,'cola_b', ST_GeomFromText('POLYGON((5 1,5 7,8 6,8 1,5 1))',-1));
INSERT INTO cola_market (mkt_id,name,geom)
VALUES (3,'cola_c', ST_GeomFromText('POLYGON((3 3,4 5,6 5,6 3,3 3))',-1));
INSERT INTO cola_market (mkt_id,name,geom)
VALUES (4,'cola_d', ST_GeomFromText('CIRCULARSTRING(6 9,10 9,6 9)',-1));
```

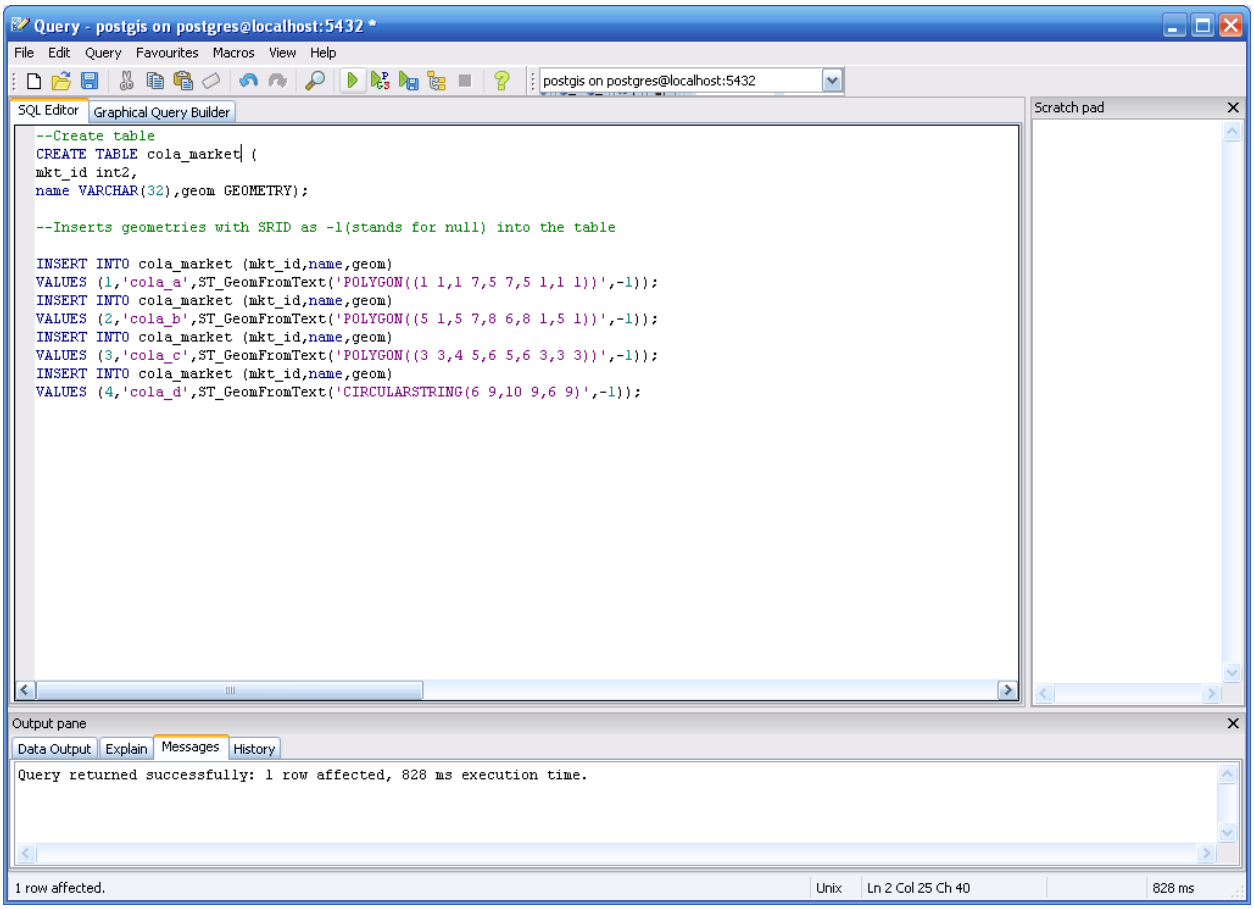
Query - postgis on postgres@localhost:5432 *
File Edit Query Favourites Macros View Help
postgis on postgres@localhost:5432
SQL Editor
Graphical Query Builder
Scratch pad
--Create table
CREATE TABLE cola_market (
mkt_id int2,
name VARCHAR(32),geom GEOMETRY);
--Inserts geometries with SRID as -1(stands for null) into the table
INSERT INTO cola_market (mkt_id,name,geom)
VALUES (1,'cola_a',ST_GeomFromText('POLYGON((1 1,1 7,5 7,5 1,1 1))',-1));
INSERT INTO cola_market (mkt_id,name,geom)
VALUES (2,'cola_b',ST_GeomFromText('POLYGON((5 1,5 7,8 6,8 1,5 1))',-1));
INSERT INTO cola_market (mkt_id,name,geom)
VALUES (3,'cola_c',ST_GeomFromText('POLYGON((3 3,4 5,6 5,6 3,3 3))',-1));
INSERT INTO cola_market (mkt_id,name,geom)
VALUES (4,'cola_d',ST_GeomFromText('CIRCULARSTRING(6 9,10 9,6 9)',-1));
Output pane
Data Output
Explain
Messages
History
Query returned successfully: 1 row affected, 828 ms execution time.
1 row affected.
Unix
Ln 2 Col 25 Ch 40
828 ms

Figure A.25. Query to create table.

Following the table creation, paste the following command in the query tool window and hit run button:

select Populate_Geometry_Columns('cola_market'::regclass);

This ensures that the geometry columns have appropriate spatial constraints and exist in the geometry_columns table.

Now, run a few spatial functions on cola_market table using the query tool:

The following function finds the union of “cola market a” and “cola market d”. Cola market d is a circle; hence we cannot apply spatial relationship functions to it. We need to convert it to line. The function should return a geometry collection object .

```
SELECT ST_AsText(ST_UNION(c_a.geom, ST_CurveToLine(c_c.geom)))
FROM cola_market c_a, cola_market c_c
WHERE c_a.name = 'cola_a' AND c_c.name = 'cola_d';
```

The following function checks if cola market a and cola market b intersect and should return true.

```
SELECT ST_INTERSECTS(c_a.geom, c_c.geom)
FROM cola_market c_a, cola_market c_c
WHERE c_a.name = 'cola_a' AND c_c.name = 'cola_b';
```

2.1.1 Examples of Well-Known Text

PostGIS geometries can be created from well-known text using ST_GeomFromText function.

- POINT(1 1)
- MULTIPOINT(1 1, 3 4, -1 3)
- LINESTRING(1 1, 2 2, 3 4)
- POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))
- MULTIPOLYGON((0 0, 0 1, 1 1, 1 0, 0 0), (5 5, 5 6, 6 6, 6 5, 5 5))
- MULTILINESTRING((1 1, 2 2, 3 4),(2 2, 3 3, 4 5))

2.2 Spatial Indexes

Indexes are extremely important for large spatial tables, because they allow queries to quickly retrieve the records they need. PostGIS is frequently used for large data sets; hence learning how to build and more importantly how to use indexes is very important. PostGIS indexes are R-Tree indexes, implemented on top of the general Gist (Generalized Search Tree) indexing schema. R-Trees organize spatial data into nesting rectangles for fast searching. Enter the following command in the query tool window to create spatial index on the geometry column “geom” of table cola_market.

```
Create Index cola_market_idx  ON cola_market USING GIST(geom);
```

Following index creation, enter the following command in the query tool and run it:

VACCUM ANALYZE cola_market;

VACUUM ANALYZE is executed to make sure that statistics are gathered about the number and distributions of values in a table and to provide the query planner with better information to make decisions around index usage.

2.3 Loading Shape Files

The data used here is in projected coordinates; the projection is “BC Albers” and is stored in the SPATIAL_REF_SYS table as SRID 3005. The shape file used is bc_pubs.

PostGIS 1.5 and above offer an easy way to load shape files using sh2pgsql-gui

- Start pgAdmin III.
- Select the PostgreSQL server under servers. Then go to Tool->connect.
- Enter the password and connect.
- Once connected select the postgis database under databases.
- Now under plugins menu select ShapeFile to PostGIS Importer.
- This will open up a window ShapeFile to PostGIS Importer as shown in Figure A.26.
- Browse to the needed shape file, enter the connection details. Also enter name of destination table , geometry column and srid.
- Click import. After successful import close the window and refresh the database. The newly imported table should appear in the list of tables.

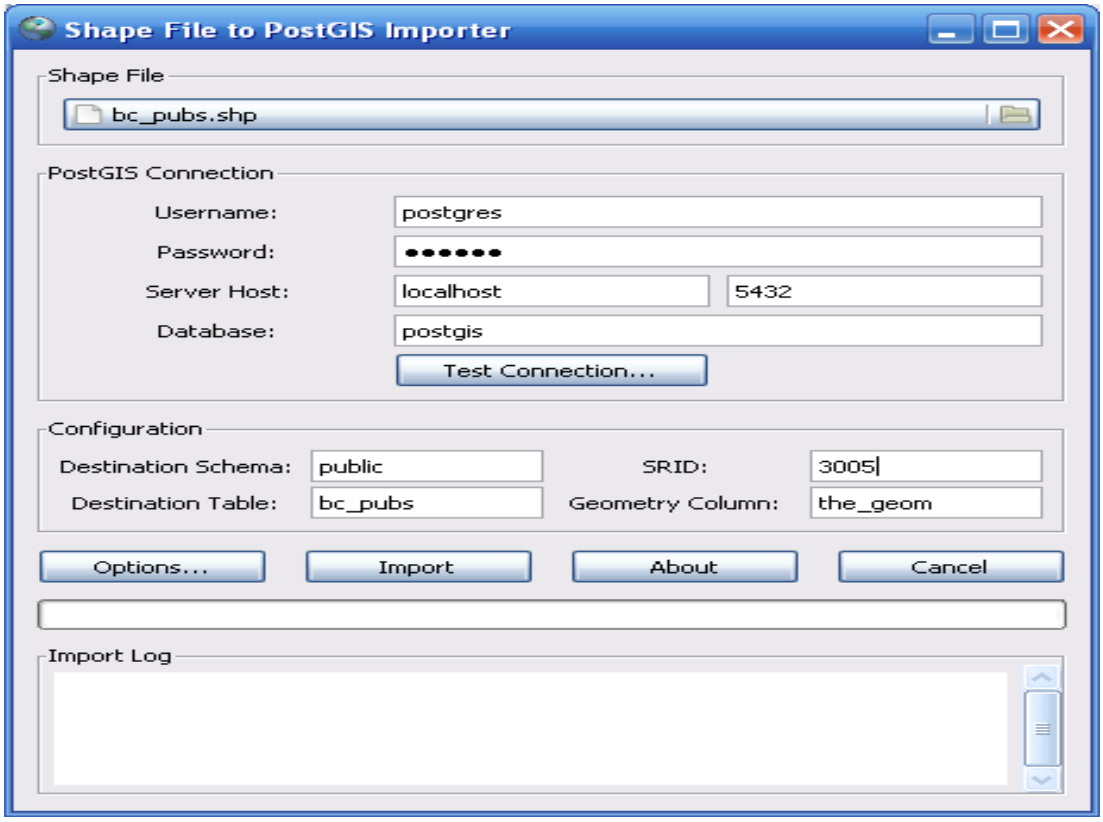

Figure A.26. Shape file to PostGIS importer.

The shape file bc_pubs has now been imported successfully.

2.4 Dispense GIS data to shape files

PostGIS allows us to create shape files out of the database tables using the pgsql2shp tool that comes packaged with PostGIS. This is a command line tool to output PostGIS spatial data to ESRI shapefile format. It outputs .shp, .shx, .dbf, .prj files. The .prj file is output only if the projection is known, fro example, if you didn't use an SRID of -1(which stands for null srid).

pgsql2shp -f filename -u username -P password databasename tablename

The following command shows the creation of shape file for bc_pubs table loaded under the postgis database .

C:\Program Files\PostgreSQL\8.4\bin>pgsql2shp -u postgres -P potato postgis bc_pubs

Initializing... Done (postgis major version: 1).

Output shape: Point

Dumping: XXXXXX [417 rows].

Note : - If the database table contains curved geometries like a circle then shape files cannot be created. The curved geometries are not yet supported by this tool

2.5 Viewing data in PostGIS

There are a number of open source options for desktop viewers / editors of PostGIS data:

- QGIS, a C++ / Qt program;
- uDig, a Java / Eclipse program; and,
- gvSIG, a Java / Swing program

Out of these, QGIS is very user friendly and provides more options.

Note - Desktop viewers for PostGIS data cannot render curved geometries.The QGIS for windows is available at http://hub.qgis.org/projects/quantum-gis/wiki/Download#Standalone-Installer-recommended-for-new-users

Note - Start the PostgreSql server by navigating to Start -> Programs -> PostgreSQL 8.4 and running “Start Server.” This needs to be done prior to running QGIS. Once QGIS is installed, go to menu Layer→Add PostGIS layer. “Add PostGIS Table(s)” window should pop up as shown in Figure A.27.

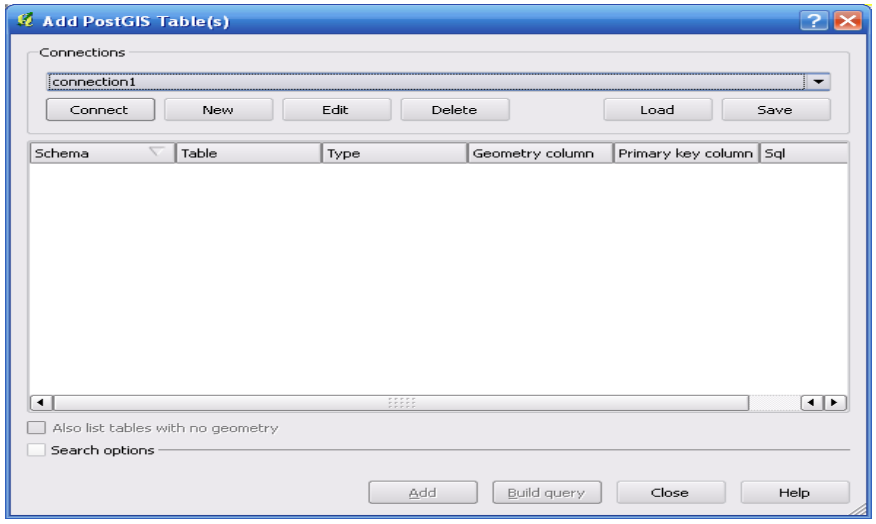

Figure A.27. Add a new connection.

Click “New” to create a new database connection.

“Create a New PostGIS connection” window should pop up as shown in Figure A.28. Enter connection name, host name (localhost), port number, database name (postgis), username (postgres) and password. Click OK .

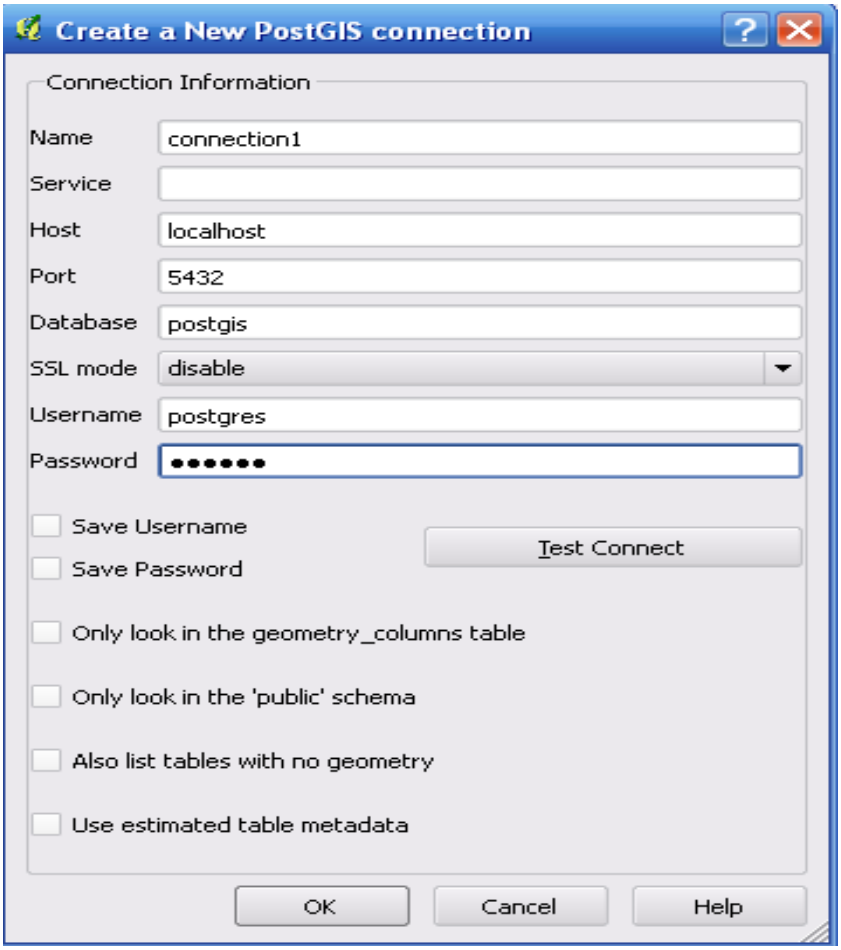

Figure A.28. Create a new PostGIS connection.

Enter Credentials window should pop up as shown in Figure A.29 Click OK.

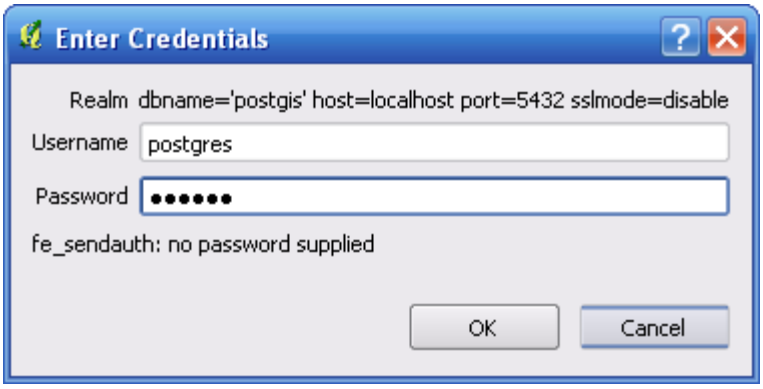

Figure A.29. Enter credentials.

It should show list of all tables in the database. Select cola_market table as shown in Figure A.30.

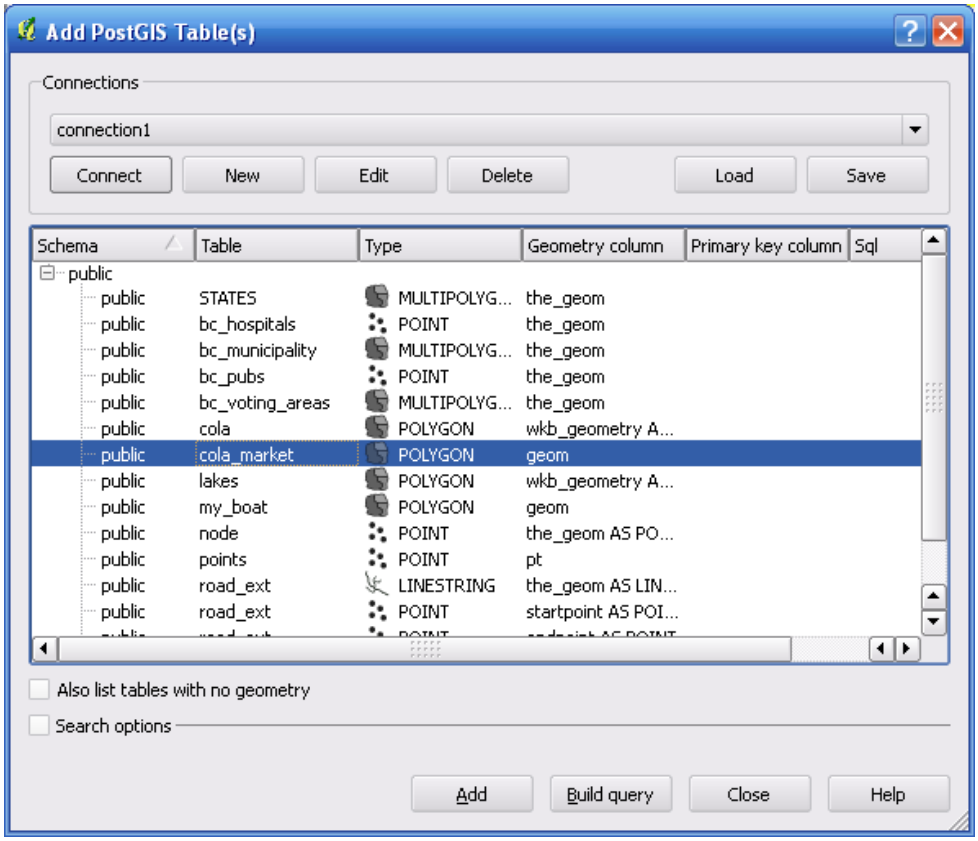

Figure A.30. Select PostGIS table.

Click “Add.”. The cola_market layer gets added as shown in Figure A.31.

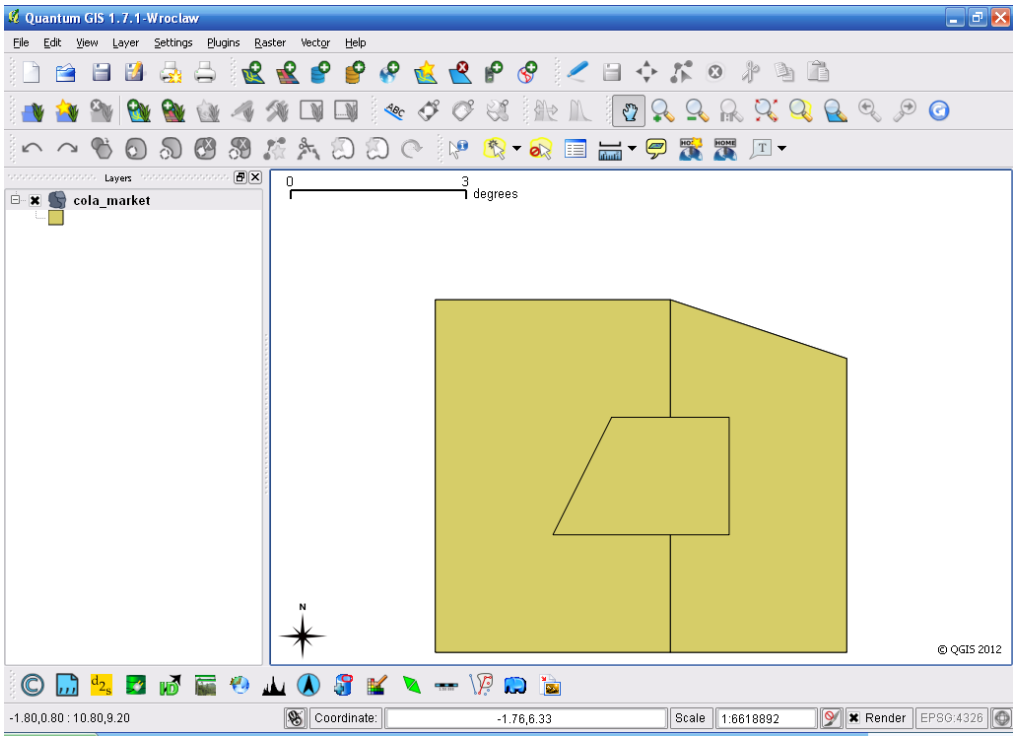

Figure A.31. cola_market table.

To view another layer, go to Menu->Layer->Add PostGIS Layer. Add PostGIS Table window appears. Click connect .A list of tables gets displayed. Select bc_pubs. Click Add. The bc_pubs layer gets added. Right click on the bc_pubs layer and select Zoom to layer extent. The layer gets displayed as shown in Figure A.32.

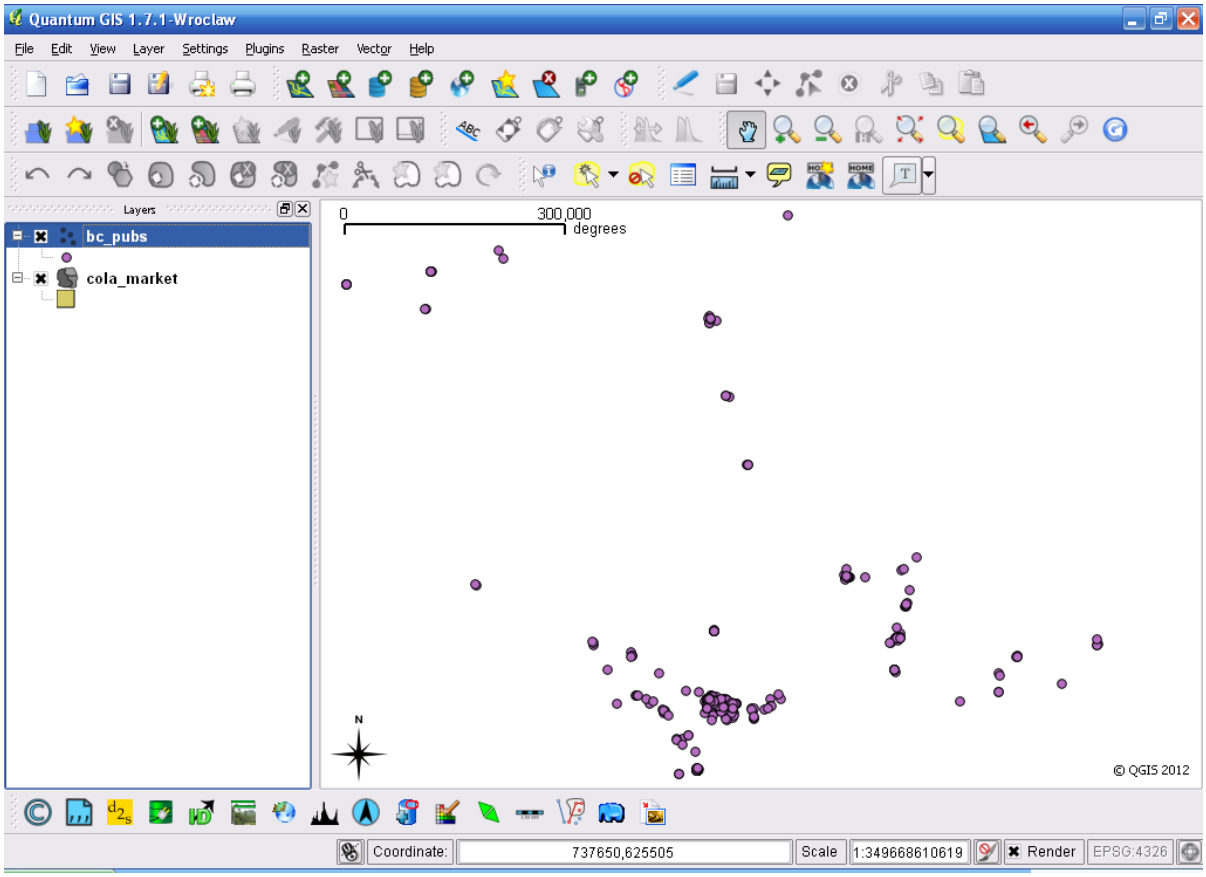

Figure A.32. bc_pubs layer.